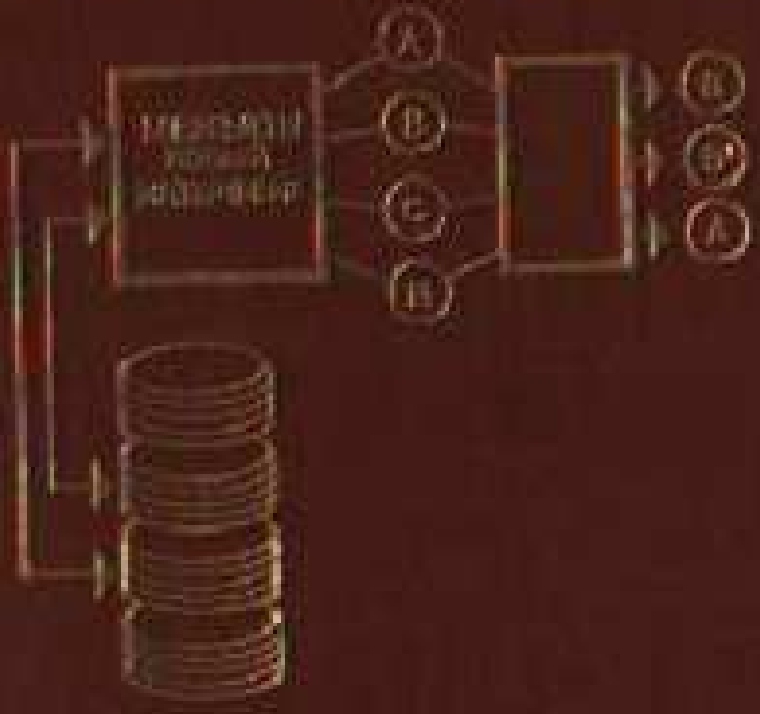
Е.М.Лаврищева
В.Н.Грибов

Сборочное программирование



АКАДЕМИЯ НАУК УКРАИНЫ
ИНСТИТУТ КИБЕРНЕТИКИ ИМ. В. М. ГЛУШКОВА

Е.М.Лаврищева
В.Н.Грищенко

СБОРОЧНОЕ ПРОГРАММИРОВАНИЕ

КИЕВ НАУКОВА ДУМКА 1991

УДК 519.681.2

Сборочное программирование / Лаврищева Е. М., Грищенко В. Н.; Отв. ред. Андон Ф. И.; АН Украины. Ин-т кибернетики им. В. М. Глушкова.— Киев: Наук. думка, 1991.— 216 с.— ISBN 5-12-002353-3.

В монографии проанализированы и систематизированы существующие подходы к сборке (посредством комплексирования и интеграции) сложных программ из более простых программных элементов (модулей и программ). Получено теоретическое обобщение и обоснование метода сборочного программирования. Определены основные его объекты и операции над ними. Рассмотрены технологические этапы жизни программного продукта согласно методу сборки и сформулированы его основные модели, методы и средства, обеспечивающие повторное использование готовых программ. Используя введенные формальные механизмы метода сборочного программирования, дана новая трактовка понятия технологии программирования, определена методика сборочного формирования программных технологий функционального типа и инженерии ведения по ним разработок прикладных программ. Рассмотрены типовые программные технологии в СОД, использующие методы контроля и оценки качества прикладных программных средств. Приведено общее описание средств автоматизации метода сборки.

Для специалистов, занимающихся созданием программных средств для объектов автоматизации на основе ЭВМ, а также студентов и аспирантов вузов соответствующих специальностей.

Ил. 40. Табл. 17. Библиогр.: с. 201 — 209 (210 назв).

Ответственный редактор *Ф. И. Андон*

Утверждено к печати ученым советом Института кибернетики им. В. М. Глушкова АН Украины

Редакция физики и кибернетики

Редактор *В. Г. Федоренко*

Л $\frac{2404010000\text{-}340}{M221(04)\text{-}91}$ 419-91

ISBN 5-12-002353-3

ПРЕДИСЛОВИЕ

Широкое распространение средств вычислительной техники в народном хозяйстве страны ставит перед специалистами важные задачи по обеспечению этой техники программными средствами (ПС) системного и прикладного характера, а также наиболее прогрессивными методами и средствами создания новых программ решения различных народно-хозяйственных задач.

Анализ потребностей народного хозяйства в ПС показывает, что их требуется более 3,5 млн наименований различного назначения и сложности. Объем выпуска в 1987 г. составил 100 млн. руб., что на порядок меньше требуемого.

Несмотря на использование потенциала всех организаций страны, состояние дел по созданию и привязке ПС к определенным задачам народного хозяйства в ближайшее время улучшится лишь частично. Это объясняется сравнительно невысоким уровнем культуры программирования (особенно разработки сложных ПС), отсутствием хорошо налаженной системы повторного использования элементов ПС, как это имеет место в машиностроении, значительным дублированием и низким качеством ПС.

Поэтому Госкомитет по вычислительной технике и информатике СССР определил на ближайшие годы программу работ в этом направлении. Основной ее целью является создание таких технологий программирования, которые обеспечили бы высокое качество разрабатываемых ПС и повышение производительности программистов в 4—6 раз к 1996 г.

Для достижения этой цели необходимо:

провести фундаментальные и прикладные научные исследования по созданию теоретических и прикладных основ индустрии разработки и производства ПС;

обеспечить организационно-методическое руководство межотраслевой кооперацией работ, направленных на создание типовых высокопроизводительных технологий разработки и производства ПС.

Одним из путей решения первой задачи является совершенствование технологии программирования в направлении перехода на промышленные методы организации разработки ПС, которые характеризуются наличием технологических процессов (ТП) и линий по производству программных продуктов согласно экономическим критериям. Основная цель программной технологии, представленной в виде ТП и технологических линий (ТЛ), — четкая

регламентация и организация работ по проектированию, разработке и изготовлению программных продуктов с высоким качеством и большой производительностью.

Повышение производительности может быть достигнуто за счет использования готовых программных элементов повторного использования, когда производственный ресурс тратится в основном на согласование интерфейсов при их объединении.

Достижение требуемого качества ПС обеспечивается за счет новых форм организации управления разработкой, предусматривающих контроль хода разработки и промежуточных состояний создаваемого опытного образца ПС.

Таким образом, новыми тенденциями в технологии программирования являются:

распространение на сферу программирования промышленных методов организации (планирование трудозатрат, определение трудоемкости, учет, контроль результатов труда и др.) ведения работ при создании ПС;

перенесение акцента с этапа программирования ПС на более ранние, начиная с анализа предметной области;

введение в практику разработки ПС таких понятий, как ТП, ТЛ, модель жизненного цикла, технологическая карта, маршрут и т. п., являющихся основным фундаментом организации управления разработкой ПС.

Наиболее быстрый переход к промышленной организации разработки может быть получен, как говорил А. П. Ершов [57], путем систематизации, формализации сборочного программирования, обеспечивающего повторную используемость готового запаса модулей — «чистой экономии труда, которая должна составлять 2/3 годового производства программных средств».

Цель настоящей монографии состоит в том, чтобы теоретически и практически определить модели, методы и средства сборочного программирования, позволяющие формально на промышленной основе осуществлять создание крупных ПС из более простых программных объектов. Монография состоит из восьми глав.

Первая глава содержит описание основных форм программирования (синтезирующее, конкретизирующее, сборочное) и постановок проблем сборочного программирования. В ней даны основные определения, сформулированы цели и задачи сборочного программирования в современном процессе создания ПС.

Во второй главе анализируются существующие методы разработки ПС сложной структуры, базирующиеся на идеях сборки (комплексирования, интеграции), определены объекты и основные модели сборочного программирования. В них входят:

модель информационного сопряжения, задающая способ преобразования экспортируемых и импортируемых данных для объектов сборки;

модель управления программными объектами, позволяющая целенаправленно управлять процессом выбора и построения программных структур из программных объектов повторно используемого и вновь разрабатываемого видов.

Отличительной особенностью описания этих моделей является применение формального аппарата для их представления, что дает возможность эффективно использовать их при программной реализации функциональных задач в СОД.

В третьей главе описываются задачи комплексирования модулей и определены формальные механизмы преобразования типов и структур импортируемых и экспортируемых данных между программными объектами в сборочном программировании.

На основе использования теории структурной организации данных К. Хоара и В. Вирта описан алгебраический подход к определению и преобразованию типов данных. Этот подход базируется на утверждениях, доказывающих необходимые и достаточные условия преобразования типов данных в классе современных языков программирования высокого уровня.

В четвертой главе сформулированы принципы выполнения операций над модульными и программными структурами и задачи построения отладочной среды сложных программных объектов. При определении типов структур программ используется математический аппарат теории графов. Использован матричный способ представления графов сложных объектов, реализуемый в виде матриц смежности и достижимости. Вводятся операции (объединения, соединения, проекции и разности) над программными объектами с модульной организацией и доказывается ряд основных положений (утверждений), определяющих действия над создаваемыми программными объектами согласно используемому матричному представлению графов.

Пятая глава посвящена методам построения интегрированных комплексов программ, применяемых на этапе проектирования и сборки. Рассмотрен ряд готовых инструментальных систем, используемых при интеграции программ. Показано использование аппарата логического проектирования интегрированных комплексов и применение экспертных систем для представления в них функциональных, программно-технических и технологических характеристик ПС, влияющих на процесс интеграции программных компонент. Рассмотрены два вида моделей (продукционная и фреймовая) для представления знаний об объединяемых программных элементах в базе знаний экспертных систем. Предложены языки описания типов данных, классов эквивалентности данных и управления программными объектами. Отмечается, что реализация этих языков на ЭВМ позволит формализовать процесс сборки интегрированных комплексов программ.

В шестой главе рассмотрены понятия технологии сборочного программирования и ее уровни, а также базисные элементы: объект разработки, методы и инструменты разработки, методы организации и управления разработкой. Проанализирован ряд систем, автоматизирующих функции предметных областей типов АСУ, СОД и сформулирована задача сборочного формирования конкретных программных технологий, реализующих задачи и функции СОД. Определены основные принципы и способы представления новых программных технологий, обеспечивающих переход на промышленные методы разработки ПС.

Седьмая глава посвящена описанию метода сборки программных технологий из объектов понятийного, технологического и инструментального типов Предложены принципы и набор правил сборочного формирования программных технологий и язык их спецификаций табличного вида. Описываются конкретные, созданные по данному методу, технологии применительно к типовым классам задач СОД.

В восьмой главе подробно рассмотрены вопросы организации и управления разработками на основе ТЛ. Описываются методы планирования трудозатрат и трудоемкости, снижения сложности и управления качеством ПС на эта-

пах жизненного цикла. Даны характеристика моделей (прогнозирующего, измерительного и оценочного типов) надежности и методика тестирования ПС. Рассмотрены основные оценочные модели надежности, базирующиеся на результатах их тестирования и испытания ПС.

Авторы считают своим долгом отметить существенный вклад академиков В. М. Глушкова и А. П. Ершова в формирование сборочного программостроения и оказавших непосредственное влияние на дальнейшее проведение работы в данном направлении. Авторы благодарят за полезные советы и замечания ответственного редактора книги Ф. И. Андона, а также сотрудников, принявших участие в обсуждении и реализации инструментально-технологических средств сборочного программирования.

СПИСОК СОКРАЩЕНИЙ

АИС — автоматизированная информационная система
АСНИ — автоматизированные системы научных исследований
АСУ — автоматизированные системы управления
АЦПУ — алфавитно-цифровое печатающее устройство
БД — база данных
БДОК — база данных объектов конструирования
БЗ — база знаний
БЗМ — библиотека загрузочных модулей
БИ — библиотека интерфейса
ВБМ — временная библиотека модулей
ВЯП — входной язык пакета
ЕС — единая серия
ЕСКД — единая система конструкторской документации
ЕСПД — единая система программной документации
ИВ — информационный вектор
ИК — интегрированный комплекс
ИС — информационная система
КТП — карта технологического процесса
ЛБ — личная библиотека
МОП — методо-ориентированный пакет
МЯИ — межъязыковый интерфейс
ОС — операционная система
ПК — программная компонента
ПО — программное обеспечение
ПП — программный продукт
ППО — прикладное программное обеспечение
ППП — пакет прикладных программ
ПС — программные средства
ПТД — проектно-технологические документы
ПрО — предметная область
РВ — разделение времени
СА — системная архитектура
САПР — системы автоматизации проектирования
СВМ — система виртуальных машин
СОД — система обработки данных

СППО— система проблемно-ориентированного обеспечения
ССПМ— система сборки программ из модулей
СУБД— система управления базами данных
ТД — технологический документ
ТЗ — техническое задание
ТП — технологический процесс
ТМ — технологический модуль
ТПР — технологическая подготовка разработки
ТЛ — технологическая линия
ТО — технологическая операция
ФА — функциональная архитектура
ФАП — фонд алгоритмов и программ
ФОТ — функционально-ориентированная технология
ЭТ — электронная таблица
ЭПД — эксплуатационно-программная документация
ЯМК — язык модульного конструирования
ЯОК — язык описания классов эквивалентности
ЯОТ — язык описания типов данных
ЯОУ — язык описания модели управления
ЯП — язык программирования
ЯПП — язык проектирования программ
ЯУЗ — язык управления заданиями

Глава 1

ПРОБЛЕМАТИКА СБОРОЧНОГО ПРОГРАММИРОВАНИЯ В ПРОЦЕССЕ СОЗДАНИЯ ПРОГРАММНЫХ СИСТЕМ

1.1. ОСНОВНОЕ СОДЕРЖАНИЕ МЕТОДА СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

Одной из главных задач современного программирования является создание теоретических и прикладных основ построения сложных программ из более простых программных элементов. Фактически эта задача является одним из способов повторного использования программных средств. А. П. Ершов [57] предложил различать три основные формы программирования на основе готовых компонент (табл. 1.1): конкретизирующее, синтезирующее и сборочное.

Конкретизирующее программирование базируется на выделении из некоторой универсальной программы отдельной ее части, настроенной на особые, определенные условия выполнения. Можно отметить два типа такого выделения. Первый характеризуется формированием конкретной программы и аналогичен процессу макрогенерации. Второй тип связан с конкретизацией информационных структур в используемом программном средстве.

При **синтезирующем** программировании строится модель программы исходя из спецификаций задачи, по которой будет синтезирована программа ее решения. Спецификация задается в терминах некоторого формального языка. На основе этой спецификации и правил построения алгоритмов из базы знаний, описывающих конкретную предметную область, происходит формирование требуемой программы.

Сборочное программирование характеризуется сборочным построением программ из готовых «деталей и узлов», которыми являются программные объекты различной степени сложности. Элементы процесса сборки присутствуют во многих методах программирования: сверху—вниз, снизу—вверх и т. д. Даже разрабатывая программы без применения каких-либо методов, программисты выделяют часто используемые операторы и оформляют их в виде отдельных подпрограмм. Возникает вопрос: в чем суть сборочного программирования, что позволяет выделить его в виде отдельного метода? Для ответа на поставленный вопрос прежде всего отметим, что сборочное программирование:

является одним из методов программирования и должно подчиняться общим закономерностям;

представляет одну из форм повторного использования программных средств;

Таблица 1.1

Форма программирования	Основа	Метод	Примеры средств автоматизации
1. Конкретизирующее программирование (идет от наличия некоторого универсального ПС)	Универсальное многопараметрическое ПС (СУБД, ППП и т. п.)	Метод адаптации, основанный на технике смешанных вычислений, оптимизационных преобразований, макрообработке и генерации ПК как конкретизации более общей программы, к конкретным условиям ее применения (привязка к структуре данных и функциям конкретной Пр0)	СКС — технология, позволяющая адаптировать СУБД к условиям применения с помощью: языка таблиц для описания типов данных и средств манипулирования ими; разбиений ПС на отдельные мелкие программы; нестандартных функций преобразования информации
2. Синтезирующее программирование (идет от постановки задачи, составляемой в виде модели вычислений или спецификации программы решения задачи)	Модель вычислений (Э. Х. Тыугу [74, 142],) отображающая для данной Пр0 понятия и отношения, задаваемые: программным модулем; моделью: описанием данных; макроопределением	Метод доказательной генерации управляющей программы, представляемой в виде последовательности операторов вызовов объектов, задающих реализацию отношений в модели вычислений	Система ПРИЗ, в которой решатель по описанию модели вычислений выбирает путь вычислений и строит программу решения задачи путем синтеза (объединения) отдельных программ, созданных для всех понятий (и их типов) задач
	Спецификация программы (В. Н. Агафонов [3, 4]) — точное описание задачи в терминах математических понятий, используемое в качестве задания на разработку программы	Метод пошагового уточнения (символический метод решения задачи) — доказательное рассуждение о сущности процесса вычисления значений данных задачи. Программа — результат этого рассуждения	Система АТЛАНТ (СО АН СССР) со встроенными типами данны позволяет создавать произвольные БД. Система СПОРА в качестве компоненты содержит сборщик программ по спецификации задачи
3. Сборочное программирование (идет от Банка ПК: модулей, программ, программных систем, КП и др.)	Схема (модель) сборки — ориентированный нагруженный граф, в вершинах которого элементы Банка, а дуги задают связи и режим функционирования	Метод комплексирования по схеме, обеспечивающий взаимосвязь объектов по управлению и по данным, преобразуемым к соответствующим типам	Система АПРОП реализует связь разноязыковых модулей, записанных в разных языках программирования ОС ЕС, посредством модулей связи, обеспечивающих необходимое преобразование типов и структур передаваемых данных

Продолжение табл. 1.1

Форма программирования	Основа	Метод	Примеры средств автоматизации
		Метод интеграции ПК по схемам и объектам из Банка, обеспечивающий создание интегрированной среды, которая включает сами ПК, программные преобразования разных типов данных и передач управления соответствующим ПК	Система ДИСУПП устанавливает связи между программными компонентами посредством маршрутной схемы

должно качественно отличаться от процессов сборки, присущих другим методам.

Жизненный цикл программного продукта в сборочном программировании — это период времени между началом разработки и завершением его использования. Он характеризуется несколькими этапами, отражающими закономерности в разработке и применении программных средств. Различные подходы, методы, инструментальные средства могут вносить отличительные особенности в отдельные этапы, однако само содержание этапов жизненного цикла программного продукта вполне определенно. Наиболее общими являются следующие этапы.

Постановка задачи. На данном этапе формулируется задача, для решения которой необходимо разработать программный продукт (ПП). Определяются целесообразность разработки, ее стоимость, сфера применения, эффект от внедрения. Условие решаемой задачи оформляется в виде ее спецификации — технического задания, постановки задачи и т. д.

Проектирование ПП. Этот этап может делиться на отдельные подэтапы, отражающие иерархическую структуру ПП,— общее проектирование, проектирование подсистем, программ, отдельных модулей. Проектирование состоит в разработке алгоритмов решения, абстрактных структур данных, операционной среды функционирования. Результаты проектирования оформляются в виде спецификаций на программные объекты и информационные структуры. На этом этапе решается также вопрос об использовании готовых программ, выполняющих требуемые функции и подфункции задачи.

Кодирование и апробирование готовых ПП. Спецификации проектирования для вновь разрабатываемых компонент ПП переводятся в тексты на языках программирования. Структуры данных конкретизируются и отображаются на реальную память. Исходные тексты переносятся на внешние носители информации. Готовые компоненты проверяются на выполнение функций на наборе исходных данных.

Тестирование и отладка ПП. Этот этап делится на несколько подэтапов, связанных с тестированием и отладкой различных видов программных объектов, входящих в состав ПП, а также с устранением различных типов ошибок, обнаруженных при кодировании или проектировании. Ошибочные результаты данного этапа приводят к выполнению предыдущих действий и необходимых корректирующих изменений в ПП.

Сборка и тестирование сложных программных объектов. Осуществляется сборка готовых ПП из числа повторно используемых и вновь разрабатываемых методом комплексирования или интеграции. В результате создается интерфейсная среда, включающая программы преобразования передаваемых данных и передач управлений соответствующим компонентам объекта сборки. Последний тестируется на множестве тестов, проверяющих созданные интерфейсы (правильность передач данных между объектами) и функционирование всего объекта. Неудовлетворительные результаты тестирования приводят к замене отдельных элементов или к исправлению в них типов и структур передаваемых данных. Эти действия вызывают выполнение указанных выше этапов жизненного цикла. Результатом данного этапа является опытный образец ПП.

Внедрение ПП. Составляется технологическая документация, разрабатываются контрольные примеры, проводится опытная эксплуатация ПП. После завершения опытной эксплуатации и исправления обнаруженных ошибок опытный образец ПП считается готовым к промышленной эксплуатации.

Сопровождение. Этот этап является последним в жизненном цикле и длится до завершения использования программного продукта. Окончание применения связано:

с разработкой более совершенного ПП;

с завершением решаемой задачи и моральным старением самого ПП или операционной среды его функционирования.

На этапе сопровождения происходит устранение ранее не обнаруженных ошибок и изменение ПП, связанные с совершенствованием технологии его применения.

Использование конкретного метода программирования сказывается на длительности каждого этапа жизненного цикла, его сложности, применяемых инструментальных средствах. В рассмотренных выше этапах жизненного цикла создаваемого ПС явно не отражены этапы его документирования и изготовления. Это связано с тем, что технология программирования регламентирует разработку документации и ее корректировку на всех этапах проектирования и разработки ПП. Поэтому к моменту внедрения предполагается, что значительная часть документации уже подготовлена. На этапе изготовления осуществляется копирование носителей программ и документации ПП. Применение инструментальных средств поддержки разработки упрощает процесс изготовления документации, и поэтому отпадает необходимость в выделении его как отдельного этапа в рамках технологии сборочного программирования.

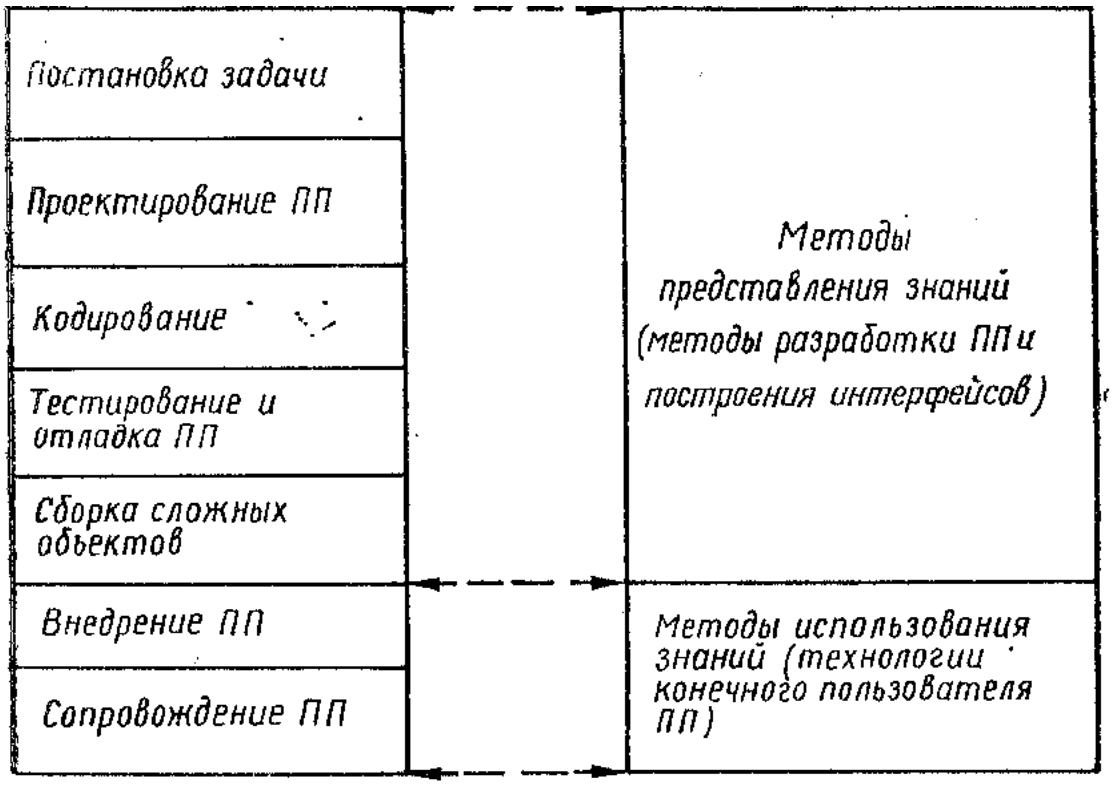


Рис. 1.1. Схема представления методов разработки и использования ПС

ПС как форма представления знаний. Программные средства фактически являются одним из способов представления знаний о решаемых задачах, конкретных предметных областях. С этой позиции рассмотрим более подробно процесс разработки и использования ПП.

При разработке баз знаний (БЗ) необходимо рассмотреть два ключевых вопроса: представление знаний и их использование. Методами представления знаний в рассматриваемой области по существу являются методы и средства разработки ПП как знания специалистов-разработчиков. При этом языками представления знаний служат языки: спецификаций, программирования, описания данных, управления заданиями и т. д. Сами разработанные ПС представляются как часть знаний о предметной области, к которой относится решаемая задача.

Использование знаний связано с этапами внедрения (частично) и сопровождения жизненного цикла ПП, т. е. его применения. Методами использования ПП являются технологии применения разработанного ПО. Результаты рассмотренного представления знаний о повторно используемых ПС отражены на рис. 1.1. В частности, рассмотренной схеме удовлетворяют все три отмеченные выше формы программирования на основе готовых компонент.

Отличие сборочного программирования от других методов. Главное отличие сборочного программирования от других методов повторного использования ПС состоит в наличии «устойчивой обратной связи», что отражено на рис. 1.2. Обратная связь означает, что в технологию использования разработанного ПП входят методы его применения при проектировании других ПС, где ПП входит как составной элемент. Термин «устойчивый» подчеркивает, что методы применения

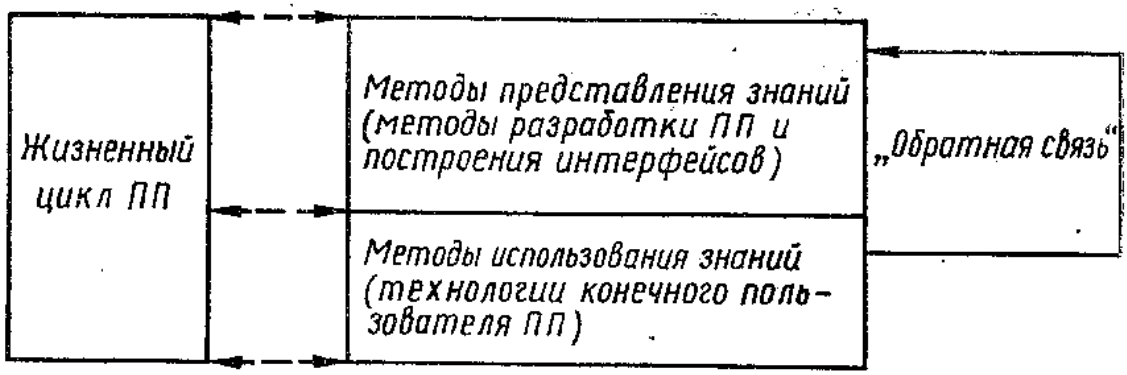


Рис. 1.2. Схема представления метода сборочного программирования

разработанных ПП в дальнейшей разработке ПС являются необходимым условием использования метода сборочного программирования.

Из рис. 1.2 следует, что метод сборочного программирования является обобщением традиционных методов разработки, так как в него дополнительно входят и методы построения интерфейсов между отдельно разработанными ПП. Соответственно изменяется и жизненный цикл. Этап проектирования включает выбор готовых ПП, которые полностью или частично решают поставленную задачу, и решение проблем интерфейсов между компонентами. Если требуется дополнительная разработка новых ПС, то формируется частная задача, решаемая с применением традиционных методов программирования. Этап кодирования упрощается (отсутствует) при наличии нескольких (всех) готовых компонент. Основное внимание при этом уделяется комплексной отладке объекта, созданного из готовых компонент.

1.2. ОСНОВНЫЕ ЗАДАЧИ СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

Процесс сборки любых изделий характеризуется: комплектующими деталями и узлами, схемой сборки (взаимосвязями отдельных компонент и правилами взаимодействия), сборочным конвейером (технологией сборки). Конкретизируем эти понятия с точки зрения метода сборочного программирования. При этом будем полагать, что используются только готовые ПП.

Комплектующие. Комплектующими в методе сборочного программирования являются программные объекты сборки различной сложности и степени готовности: модули, сегменты, программы, комплексы программ, пакеты прикладных программ и т. д. Результатом сборки может быть любой программный объект, за исключением модуля. Однако для практического применения выбирается несколько базовых типов программных объектов, рассматривающихся как компоненты, и результат использования метода сборочного программирования.

Для технологичности сборки все объекты должны иметь паспорта, содержащие данные, необходимые для информационного сопряжения и организации совместного функционирования в рамках программного объекта более сложной структуры. Важное условие сборки состоит в наличии большого числа разнообразных комплектующих, т. е. ПС,

обеспечивающих решение широкого спектра задач из различных предметных областей.

Схема сборки. Под схемой сборки программных объектов понимается схема их взаимодействия, определяющаяся непосредственными обращениями к компонентам (типа оператора CALL) или последовательностью их выполнения. При этом взаимодействие каждой пары объектов зависит от совместного использования данных. В общем случае схема сборки состоит из совокупности моделей, отражающих различные типы связей между компонентами: передача управления, обмен информацией, условия совместного функционирования и т. д.

Операции сборки выполняются согласно паспортам объектов и правилам сопряжения. В соответствии с этим информация в паспортах должна быть систематизирована и выделена в отдельные группы согласно ее функциональному назначению: передаваемые данные и их типы, вызываемые объекты, совместно используемые файлы и т. д. Правила сопряжения определяют совместимость объединяемых компонент и содержат описание функций, необходимых для согласования различных характеристик программных объектов, отраженных в их паспортах.

Сборочный конвейер. Процесс сборки может производиться ручным, автоматизированным и автоматическим способами. Как правило, последний способ практически неосуществим, что связано с отсутствием формального определения и представления программных объектов. Ручной способ нецелесообразен, так как сборка готовых компонент представляет собой большой объем действий, носящих скорее рутинный, чем творческий, характер. Наиболее приемлемым оказывается автоматизированный способ сборки. При этом сборкой управляет определенная процедура (система), которая по заданным спецификациям (моделям) программ осуществляет сборку на основе стандартных правил сопряжения под управлением человека. Средства, поддерживающие данный способ сборки, называются инструментальными средствами сборочного программирования. К ним относятся средства комплексирования (объединения компонент в более сложный объект); средства интерфейсов (реализации сопряжения объектов согласно их паспортам и стандартным правилам сопряжения); средства описания и использования моделей сборочного программирования (совокупность моделей сборки различных программных объектов).

Из рассмотренной схемы сборки следует выделить задачи сборочного программирования и условия его применения:

выбор компонент из числа готовых программных объектов для решения класса задач из определенной предметной области;

проектирование структуры (моделей) создаваемого объекта, элементами которого являются готовые программные объекты, определенные на множестве данных выбранной предметной области;

сборка, согласно моделям, программного продукта, реализующего функции, соответствующие целям и задачам автоматизируемой предметной области.

Необходимые условия применения метода сборочного программирования включают:

наличие большого числа разнообразных программных продуктов, являющихся компонентами сборки;

паспортизация программных объектов сборки;

наличие достаточно полного набора стандартных правил сопряжения и алгоритмов, их реализующих;

наличие средств автоматизации процесса сборки;

создание технологий применения разработанных объектов для использования в более сложных программных продуктах.

Последнее условие означает, что должны существовать определенные формы представления ПС как знаний о предметных областях, универсальные с точки зрения проектирования и разработки ПО.

1.3. ФОРМЫ ПРЕДСТАВЛЕНИЯ ПС КАК ЗНАНИЙ О ПРЕДМЕТНЫХ ОБЛАСТЯХ

В настоящее время сложились устойчивые понятия о методах представления и использования знаний: представление продукционными правилами; фреймовое представление; семантические сети; представление знаний с помощью исчисления логики предикатов; представление нечетких знаний. Очевидно, что данные методы неприменимы для сборочного программирования несмотря на то, что существуют языки представления знаний, которые одновременно являются и языками программирования (Пролог, Лисп и др.)

Будем исходить из того факта, что знаниями о предметной области для сборочного программирования служат разработанные ПС. Вернемся к промышленному подходу к сборке. Из заготовок формируются отдельные детали, из деталей — мелкие узлы, из мелких узлов — более крупные и так до полного изготовления изделий. На каждом этапе составляется конструкторская документация, включающая документацию на комплектующие сборки и на результирующий продукт. Аналогичный подход используется и в сборочном программировании. Формой представления знаний элементарных объектов являются сами ПС. Формой представления знаний результата сборки служат знания об отдельных объектах и знания, приобретенные в процессе сборки. Последние включают дополнительные знания о предметных областях (отражение главной цели разработки ПП в рамках сборочного программирования) и о самих ПП (схемы сборки, программные интерфейсы).

Таким образом, применение сборочного программирования позволяет получить новые знания не только о предметных областях, но и о процессе разработки ПО. Использование метода расширяет сферу его применения. Это является еще одним отличием от традиционных методов разработки ПО, главная цель которых заключается в приобретении или накоплении новых знаний о предметных областях.

Возвратимся к формам представления ПС как знаний. Знания об элементарных объектах сборки включают спецификации, исходные тексты на языках программирования, описания данных (при работе с внешними файлами и базами данных), процедуры на языках управления заданиями для вызова ПП и т. д. Дополнительные знания, полученные в процессе сборки, представляются в виде совокупности

моделей и программных интерфейсов сопряжения отдельных компонент.

Автоматизированный подход к сборке ПП позволяет упорядочить процесс приобретения новых знаний на основе применения специальных средств описания и обработки моделей, а также генерации интерфейсов согласно набору стандартных правил. Эти средства и являются основой создания технологий применения разработанных ПП для использования в более сложных программных продуктах. Каждый ПП в рамках сборочного программирования (с соответствующими средствами автоматизации) создается на единой методологической основе и без существенных изменений может использоваться в других процессах сборки.

В данной главе метод сборочного программирования рассмотрен на концептуальном уровне без конкретизации его составляющих. В остальных главах уточняется его содержание применительно к конкретным видам объектов, правил сопряжения, моделей, описывается несколько практических реализаций и приводятся рекомендации по применению и совершенствованию самого метода.

Глава 2

ОБЩИЕ ВОПРОСЫ РЕАЛИЗАЦИИ МЕТОДА СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

2.1. МЕТОДЫ КОМПЛЕКСИРОВАНИЯ ПРОГРАММНЫХ ОБЪЕКТОВ

Из анализа существующих методов разработки ПО [1, 20, 33, 65, 66, 70, 147] следует, что интерфейсам для комплексирования программных объектов как самостоятельным инструментальным средствам уделяется недостаточное внимание. Предполагается, что все вопросы комплексирования решаются на этапе проектирования ПО, что не всегда согласуется с практическими результатами. Задачи сопряжения программных объектов решаются различными методами и с помощью различных средств. Можно выделить следующие группы методов решения задач сопряжения:

1) комплексирование на основе аппаратных средств ЭВМ;
2) изменение структуры исходных текстов модулей;
3) использование средств компиляторов и редакторов связей для сопряжения модулей;
4) использование промежуточных языков систем программирования;
5) организация связи с помощью управляющей программы;
6) использование при комплексировании специальных модулей связи программных объектов;
7) системное комплексирование готовых программных объектов.

Рассмотрим эти методы более подробно.

1. Комплексирование с использованием аппаратных средств ЭВМ основано на специальной системе команд и способе представления данных. Архитектура таких ЭВМ отлична от модели фон Неймана. Оперативная память позволяет хранить различные типы данных, при этом каждая ячейка содержит специальные признаки (тэги, дескрипторы и т. д.), позволяющие правильно интерпретировать их значения и выбирать соответствующие режимы обработки. Система команд, учитывающая эти признаки [5], позволяет не рассматривать задачи, связанные с преобразованием внутреннего представления данных в модулях.

В более совершенных ЭВМ, таких, как система Интел 432 [118], ориентированная на использование конструкций языка программирования Ада, дескрипторы существуют не только для простых типов

данных, но и для сложных. В ее архитектуру входят специальные команды:

CREATE OBJECT (создать объект).

CREATE TYPE OBJECT (создать типизированный объект),

При выполнении этих команд создаются как обычные объекты, так и объекты «расширенного типа». Типы объектов проверяются на этапе трансляции и при выполнении команд, использующих эти объекты.

В отличие от системы Интел 432 архитектура системы В-1700 фирмы BURROUGHS ориентирована на несколько языков программирования — Бейсик, Фортран, Кобол, РПГ/2 [118]. Система динамически меняет свою архитектуру во время диспетчеризации различных прикладных программ. При выполнении программы аппаратными средствами осуществляется переключение микропрограмм на режим ЯП данной программы. Аналогичным образом происходит обработка данных.

Среди отечественных разработок следует отметить МВК «Эльбрус», архитектура которого ориентирована на язык высокого уровня [130]. Процесс комплексирования в этом комплексе имеет следующие особенности [54]. Преобразование относительных адресов рабочих программ в абсолютные осуществляется на аппаратном уровне. Необходимость преобразования типов данных отсутствует за счет введения тэгов и возможности выбора алгоритма обработки данных на аппаратном уровне. Основная задача Комплексатора (системной программы объединения модулей) состоит в добавлении новых процедур в базовую программу и/или замене существующих новыми версиями.

2. При комплексировании, основанном на изменении структуры и исходного текста модулей, проблемы сопряжения решаются путем внесения дополнительных операторов и команд в исходный текст для каждой пары взаимодействующих модулей в соответствии с подходами [15, 46, 47, 112, 119, 145]. Основное достоинство данного метода состоит в том, что разработчик вносит наиболее оптимальные (по времени выполнения или объему дополнительной памяти) изменения. Любой метод, являющийся универсальным, потребует больше ресурсов, но и имеет следующие недостатки: разработчик должен самостоятельно осуществлять все вопросы комплексирования, особенности передачи управления, структуры передаваемых данных; объем работ по разработке программных объектов увеличивается, так как требуется дополнительное время на написание и отладку изменений, вносимых в отдельные модули; модули являются зависимыми от своего окружения, их замена и/или включение в другие объекты являются сложной задачей; при разработке модулей на ЯП высокого уровня не все проблемы сопряжения могут быть решены одними языковыми средствами— часть проблем разработчик должен реализовывать на языке типа Ассемблера.

Рассматриваемый метод комплексирования целесообразно применять при объединении небольшого числа модулей, когда использование средств универсального интерфейса не оправдано.

3. Применение содержащихся в компиляторах средств для связи модулей также связано с изменениями в исходных текстах. Однако по сравнению с предыдущим методом изменения эти незначительные. Разработчик только указывает место в модуле и дополнительные средства, включаемые в это место, для вызова другого модуля. Примером рассматриваемого подхода может служить комплексирование с применением оптимизирующего транслятора с языка ПЛ/1 ОС ЕС [110]. При вызове модуля, написанного на ЯП Фортран, из ПЛ-модуля последний будет содержать следующие дополнительные операторы:

```
DCL FOR OPTIONS (FORTRAN)
    ENTRY        ((описания типов данных передаваемых
                  параметров модулю FOR));
    операторы ЯП
    CALL FOR     ((список передаваемых параметров)).
```

При вызове ПЛ-модуля из Фортран-модуля его заголовок будет иметь вид

```
PL: PROC ((список параметров)) OPTIONS (FORTRAN).
```

Аналогичным образом осуществляется связь с модулем, написанным на языке Ассемблер. В этом случае вместо параметра FORTRAN используется параметр ASSEMBLER.

Реализацию аналогичного подхода к комплексированию объектов также предусматривают трансляторы с ЯП Паскаль в операционных системах ОС РВ и РАФОС для СМ ЭВМ при обращении к Фортран-модулям.

К достоинствам данного метода комплексирования следует отнести незначительные дополнительные затраты на внесение изменений в исходный текст и освобождение разработчика от знаний особенностей сопряжения. Основные его недостатки состоят в следующем: модули зависят от своего окружения (в рассмотренном примере это ПЛ-модули); невозможно полностью отделить процесс трансляции модулей от процесса комплексирования; не все трансляторы обеспечивают возможности по сопряжению модулей.

Последний недостаток довольно существенный, так как требует применения других методов комплексирования.

4. Комплексирование на основе использования промежуточных языков систем программирования является следствием применения специальных промежуточных кодов трансляции, называемых Р-кодами [146]. Входными языками трансляторов в таких системах служат обычные языки программирования высокого уровня. Благодаря применению единого промежуточного Р-кода достигается совместимость объектных программ. Наиболее известными промежуточными кодами в настоящее время являются UCSD-код, Р-4, М-код. Система UCSD поддерживает программирование на ЯП Паскаль, Фортран-77, Бейсик, АПЛ, Лисп, Модула-2 [207]. Разработаны специальные компьютеры, внутренними языками которых являются UCSD-код [44, 162] и М-код [208].

Система UCSD представляет собой абстрактную машину, а система команд ее ориентирована на работу со стеком. Общее число команд около 300. Стек используется для хранения общих и локальных данных, включая параметры процедур и временные переменные, а также коды процедур активных сегментов. Отдельный сегмент процедур представляет собой объект, загружаемый целиком в память из внешнего устройства. В момент вызова процедуры создается ее сегмент данных, состоящий из параметров, локальных данных и маркера, определяющего начало области стека для вызванной процедуры. Проверка совместимости типов проводится на этапе компиляции.

К достоинствам подобных систем относятся: простота подхода к комплексированию; отсутствие специфики сопряжения и высокая степень переносимости ПО. Главным недостатком применения Р-кодов является снижение скорости выполнения программы, так как она выполняется в режиме интерпретации. Использование ЭВМ, внутренним языком которых являются Р-коды [44, 208], устраняет этот недостаток.

5. Организация связи с помощью управляющей программы требует наличия универсальной программы сопряжения. Взаимодействие между парой модулей не прямое, а осуществляется через управляющую программу, которая выполняет все необходимые операции сопряжения для данной пары. Пример такого подхода приведен в [124]. Для взаимодействия модулей, написанных на языках Фортран, ПЛ/1, Ассемблер, разработана управляющая программа, содержащая 6 точек входа. Каждая соответствует определенному типу взаимодействия модулей. Необходимо отметить, что данная программа позволяет решать вопросы сопряжения, связанные с передачами управления, не затрагивая преобразования передаваемых данных.

К достоинствам метода комплексирования следует отнести наличие готовых средств для реализации интерфейса (управляющая программа), единый подход к решению вопросов сопряжения и незначительные усилия для освоения управляющей программы, требуемые от разработчика проблемных модулей. К недостаткам — наличие готовой управляющей программы, которая могла бы решать все задачи сопряжения; необходимость изменения модульной структуры программы и зависимость модулей от окружения, проявляющаяся в привязке к конкретным точкам входа при вызове других модулей.

6. Метод комплексирования, основанный на применении специальных модулей сопряжения, имеет много общего с методом использования управляющей программы. Взаимодействие между модулями не прямое, а осуществляется через специальный модуль сопряжения, называемый модулем связи. Модуль связи разрабатывается для каждой пары взаимодействующих модулей в программном комплексе. Данный метод основывается на библиотеке интерфейса (БИ), содержащей макроопределения и загрузочные модули. Каждое макроопределение выполняет определенную функцию сопряжения, связанную с передачей управления и данных между объединяемыми модулями. Последовательность макрокоманд, составленная согласно используемому подходу, образует модуль связи и включается в состав загрузоч-

ного агрегата для каждой пары разноязыковых объектов. Модуль связи состоит из:

оператора идентификации имени модуля связи;

макрокоманды пролога, включающей подготовительные операции для выполнения модуля связи;

последовательности макрокоманд, обеспечивающей преобразование передаваемых данных согласно форматам, определенным в вызываемом модуле;

последовательности макрокоманд, обеспечивающей создание функциональной среды вызываемого модуля и передачу ему управления;

последовательности макрокоманд, обеспечивающей обратное преобразование возвращаемых вызывающему модулю данных;

макрокоманды эпилога модуля связи, обеспечивающей возврат управления вызывающему модулю;

оператора завершения модуля связи.

К достоинствам данного подхода следует отнести: освобождение разработчика от знаний особенностей сопряжения; наличие программных заготовок в библиотеке интерфейса для построения интерфейсного промежуточного модуля связи; наибольшую степень независимости модулей от окружения по сравнению с другими методами комплексирования; возможность реализации новых функций сопряжения за счет расширения БИ новыми средствами преобразования данных между объединяемыми объектами; к недостаткам — довольно большой объем работ, связанный с программированием модулей связи (даже с применением макрокоманд); значительное увеличение числа модулей, включаемых в программный агрегат, что усложняет управление модульными структурами.

Эти недостатки в значительной степени устраняются при автоматизированном подходе к комплексированию Пример реализации такого подхода описан в [89—96].

7. Метод комплексирования готовых программных компонент существенно отличается от ранее рассмотренных тем, что в нем переход к новой программной компоненте осуществляется не стандартным механизмом вызова, а с помощью средств операционной системы после завершения работы предыдущей компоненты. Применение данного подхода описано в [52]. Механизм вызова представляется на языке управления заданиями (ЯУЗ) ОС ЕС в виде пакета, задания состоящего из нескольких шагов. Первые шаги описывают обращения к прикладным программам, а на последнем происходит вызов специальной программы, осуществляющей обращение к программе системного ввода ОС ЕС (RDR). Выбор нужной компоненты состоит в модификации пакета задания на ЯУЗ (изменение имени программы и описания ресурсов) и повторном обращении к программе системного ввода.

К достоинствам данного метода следует отнести простоту использования средств комплексирования, полную независимость программных компонент друг от друга и минимум знаний о процессе комплексирования.

Самым существенным недостатком является то,что все программные объекты должны быть доведены до уровня готовности к выполнению

(т. е. пройти этапы трансляции, редактирования связей, занесения в библиотеку загрузочных программ), а замена программных объектов описанным способом требует значительных вычислительных ресурсов, особенно времени выполнения.

Таблица 2.1

Метод комплексирования	Выбор архитектуры ЭВМ	Разработка модулей	Разработка дополнительных ПС	Трансляция модулей	Сборка модулей	Выполнение ПП	Возможность автоматизации
1. Использование аппаратных средств	~				∨	ṽ	+
2. Изменение структуры и текста модулей		~	ṽ		∨		—
3. Использование средств компиляторов				~	∨		+—
4. Использование промежуточных языков	~			~	∨	ṽ	+
5. Использование управляющей программы	ṽ				∨		+—
6. Использование модулей связи	~		~		∨		+
7. Комплексирование готовых программ						∨	+

Обобщенные результаты анализа методов комплексирования на разных стадиях разработки программных комплексов представлены в табл. 2.1. Здесь приняты следующие обозначения:

«~» — отметка о необходимости решения задачи сопряжения пар модулей (по управлению и по данным);

«∨» — отметка о необходимости решения проблемы объединения программных компонент в единый агрегат;

«+» — возможность автоматизации данного метода комплексирования, а «—» — ее отсутствие;

«+ —» — возможность автоматизации частных задач и отсутствие общих путей автоматизации.

Вопросы автоматизации построения интерфейсов тесно связаны с задачей повторного использования программных средств, которой в настоящее время уделяется большое внимание. В работе [131] предлагается язык LIL (Library Interconnection Language) для использования компонент, хранящихся в библиотеках общего пользования. Эффективность такого подхода подтверждает опыт японских фабрик разработки ПО, где производительность труда программистов за год при использовании ранее разработанных программных средств возросла на 20 % [209, 210].

Необходимость повторного использования программных средств привела к созданию метода сборочного программирования [57, 131, 132]. В этом случае основные затраты на разработку связаны с созданием новых программных компонент и комплексирование их со ста-

рыми. В [142] приводится описание инструментальных средств для разработки программных компонент многоразового использования. В качестве иллюстрации выбран класс задачи обработки данных.

2.2. СРЕДСТВА АВТОМАТИЗАЦИИ МЕТОДОВ КОМПЛЕКСИРОВАНИЯ

На базе описанных выше методов комплексирования создано множество систем автоматизации программирования и комплексирования, разработаны специальные языки и технологии. Необходимо отметить, что в большинстве подходов с понятием комплексирования свызывают и синтез программ. В общем случае это справедливо, так как в процессе комплексирования есть множество черт, сходных с процессом синтеза, и наоборот.

В частном случае для систем модульного программирования эти процессы имеют определенные границы. Ниже приведен обзор известных методов и систем синтеза и комплексирования программных средств.

1. В [131, 132] рассмотрен метод композиционного программирования. Суть его состоит в уточнении понятия семантической структуры программ. С этой точки зрения данные уточняются как именные данные, запрашивающие программы (выражения и операторы ЯП с семантической точки зрения также рассматриваются как запрашивающие программы) — как именные функции. Средства конструирования запрашивающих программ рассматриваются как композиции именных функций. Программы отождествляются с выводами в универсальной программной логике, роль аксиом в которой выполняют именные функции, а роль правил вывода — композиции.

2. Метод концептуального программирования рассмотрен в [142]. Для конкретной предметной области описываются понятия, достаточные для выражения смысла решаемых задач. В терминах этих понятий выполняется описание каждой задачи, по которому осуществляется автоматический синтез программы и выполняются вычисления. Алгоритмы структурного синтеза программ [75] основаны на доказательстве существования решения, которое находится программой. Алгоритмы поиска рассматриваются для трех различных классов задач — линейной структуры, ветвящейся структуры и структуры с подпрограммами. Проблемы интерфейсов решаются на этапе синтеза программ. Большинство принципов, положенных в основу концептуального программирования, реализовано в системе ПРИЗ [74]. Развитием ее явилась мобильная инструментальная система, предназначенная для автоматизации построения пакетов прикладных программ [59].

3. Системой, близкой к ПРИЗ по своим концепциям синтеза программ, является СПОРА [87]. В ней синтез основан на использовании баз знаний. Система обеспечивает построение программы по спецификации задачи. Сам процесс построения представляется как получение доказательства теоремы существования решения задачи в некоторой формальной теории. Архитектура базы знаний ориентирована на модульный принцип построения системы в целом.

4. Целый ряд систем автоматизации программирования построен на принципе использования промежуточных языков. В качестве примера можно привести язык АЛМО [75], систему СИМПР [58, 134], язык MESA [206] и др. Особый интерес представляет СИМПР, являющаяся многоязыковой системой модульного программирования. В основу промежуточного языка положен абстрактный семантический уровень интерпретации программ, который содержит понятия, необходимые для описания внешних свойств модулей. В качестве формы записи языка используется польская инверсная запись. Слогами этой записи являются операции языка или указатели объектов транслируемой программы, к которым применяются данные операции. Абстрактный семантический уровень открыт для расширения, что позволяет включать в систему новые языки программирования. Включение нового языка — это фиксация всех языковых понятий, которые могут выступать в качестве внешних свойств модуля, в терминах уровня интерпретации программ. Интерфейс между разноязыковыми модулями не требуется, так как внешние свойства каждого модуля определены на едином для системы СИМПР абстрактном семантическом уровне.

Язык MESA предназначен для создания больших программ по модульному принципу. Он ориентирован на использование ЯП с блочной структурой типа Алгол, Паскаль и др.

5. Подход, связанный со структурной интерпретацией языков высокого уровня, нашел применение в R-технологии, описанной в [30, 150]. При программировании в терминах специального R-языка описывается структура исходной предметной области. Затем проектируется логическая структура данных, связанная с соответствующими алгоритмами обработки. В настоящее время ведутся работы по визуализации R-языка и распространению графического стиля программирования, основанного на использовании концепции R-языка, на современные языки высокого уровня (МОДУЛА-2, Си, Паскаль и др.).

Практически во всех системах автоматизации программирования приходится решать задачи комплексирования программных компонент и управления модульными структурами программ.

6. В [148] описана программная система BPS, предназначенная для разработки больших программных комплексов. Входными языками являются Ассемблер и специально разработанный язык BPS/L. За основу разработки BPS/L был взят язык МОДУЛА, дополненный специальными инструкциями, предоставляющими необходимую информацию для комплексирования. В состав этой информации входят: список модулей, с которыми связан данный модуль, описание объектов, используемых вне модуля, описание локальных объектов и т. д. Специальные средства связи обеспечивают построение графа программного агрегата и управление комплексированием на основе указанной информации. При программировании модуля с использованием языка Ассемблера его структура дополняется специальными инструкциями, аналогичными таковым в языке BPS/L.

7. В основу системы автоматизации модульного программирования (САПМ) [134] положен принцип инверсного преобразования гра-

фа достижения заданной цели. Для множества пакетов прикладных программ (ППП) строятся орграфы предметных областей. Для орграфов задается совокупность матриц связей, описывающих взаимосвязи программных модулей. Каждой вершине графов ставится в соответствие цель, которую необходимо достигнуть, например номер задачи, выполнение которых обеспечит достижение данной цели. Для решения задачи на графе указываются конечные цели и с помощью матриц связи строятся обратные цепочки с фиксацией промежуточных целей. Вопросы интерфейсов между модулями не рассматриваются.

8. В работе [150] излагается методика проектирования модульных программных структур, которая заключается в преобразовании программной структуры в модульную. Определены 4 типа модулей: функциональный; модуль данных; абстрактная структура данных; абстрактный тип данных. Каждый модуль содержит информацию, которая описывает его внешнее поведение. Все связи осуществляются через модульный интерфейс, проведена классификация видов связей модулей, разработаны объективные критерии создания модульных структур и методика их использования. Методология проверена практически, однако теоретическая сторона нуждается в обосновании.

9. В инструментальном комплексе ДИСУППП [125] комплексирование модулей осуществляется на основе многоуровневой графовой модели предметной области. Сами функциональные модули реализуют отдельные шаги вычислений и хранятся в специальной базе. Комплекс ДИСУППП эффективно применяется при конструировании диалоговых систем.

10. В рамках системы SARA [186] разработан язык модульного взаимодействия MIL [203]. Язык обеспечивает описание взаимодействия не только между программными модулями, но и между программным модулем и модулем данных, примером которого может служить файл. В последнем случае взаимодействие эквивалентно операциям обмена. С помощью средств языка MIL описываются как сами модули, так и интерфейсы между ними.

11. Описание взаимодействия модулей используется в языке мультимодульного программирования (ММП), являющегося составной частью алгоритмического языка Маяк [116]. Программа в ММП представляется как сеть алгоритмических модулей переменной структуры. Язык позволяет описывать динамические взаимодействия между модулями и обеспечивает возможность параллельных вычислений.

Другими языками и системами, обладающими аналогичными возможностями, являются DREAM [207], GYPSY [206], ADAPT, PROTEL, CDL2, Candalf [81, 122].

2.3. ОБЪЕКТЫ СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

Построение любой формальной теории или системы основано на определении объектов, рассматриваемых этой теорией, их характеристик и свойств, а также на выборе операций над данными объектами. Следуя этой схеме, необходимо определить объекты сборочного про-

граммирования, понимая под операциями различные функции комплексирования — информационное и программное сопряжение, обеспечение интерфейса по управлению, создание интегрированных сред, языков общения с пользователями и т. д.

Строгих формальных определений программных объектов в настоящее время не существует. В практике программирования встречаются такие термины, как модуль (включая подпрограмму, функцию, процедуру, программную секцию), макромодуль, программа, программный сегмент, комплекс программ, система, подсистема и т. д. Каждое понятие в контексте избранных вопросов исследований может иметь тот или иной смысл, что связано с выбором конкретных концепций, методов, моделей для анализа. На практике указанные термины употребляются для сходных по признакам программных объектов, интуитивно понятных большинству специалистов. Из этого следует, что объекты, их свойства, операции над ними, рассматриваемые в данной книге, будут анализироваться только с позиций сборочного программирования и полученные результаты не претендуют на всеобщность применения.

Рассмотрим два типа программных объектов сборочного программирования. К первому отнесем объекты, которые при разработке ПО могут менять формы своего представления. В частности, к ним относятся макромодуль, модуль, программный сегмент. Все эти объекты изменяют форму представления (исходную, объектную, загрузочную) на различных этапах разработки ПО. Макромодуль используется на этапе макрогенерации, результатом которой является исходный или объектный текст модуля. Модуль может проходить этапы трансляции и сборки (редактирования связей). Программный сегмент включается в более сложный объект на этапе редактирования связей.Среди объектов первого типа выберем базовый—модуль. Все проблемы комплексирования для объектов данного типа сводятся к проблемам взаимодействия модулей и построения из них более сложных программных структур.

Ко второму типу относятся программные объекты, не меняющие форму представления при их использовании: программы, комплексы, системы и т. д. Все эти объекты автономны, решают определенные классы задач, и для их использования не требуется дополнительная обработка. Базовым объектом этого типа принимается программа. Предполагается, что более сложные объекты — комплексы, системы и т. д. — представляют собой множество взаимодействующих программ. Сама программа как конечный результат процесса разработки ПО состоит из некоторого множества модулей.

Определение понятий *модуль* и *программа* здесь не формализовано. Более строгое определение этих понятий в рамках метода сборочного программирования будет приведено ниже.

В программных системах новых поколений в качестве объектов используются пакеты прикладных программ, а любая система обработки данных представляется как открытая система ППП. Сюда относится архитектура ISA и SAA.

Выбор модуля и программы в качестве базовых программных

объектов сборочного программирования оказывает принципиальное влияние на реализацию функций комплексирования, что вызвано следующими особенностями этих объектов.

1. Возможность изменения объектов при реализации интерфейсов между ними. Для модулей такие изменения допустимы. К ним, в частности, относятся изменения в описаниях данных и порядке следования передаваемых параметров, особые режимы трансляции и редактирования связей. Программы такой возможности не имеют. Поэтому если программа формирует файл определенной структуры, то для изменения последней необходимо вносить изменения в модули, входящие в программу, что противоречит требованию неизменяемости формы представления.

2. Уровень реализации интерфейсных функций. Для реализации интерфейсов между модулями используются механизмы, определяемые ЯП и соответствующими трансляторами. Для информационного сопряжения необходимо наличие средств преобразования данных (типов и внутреннего представления) ЯП. Решение проблемы связи модулей зависит от особенностей механизмов управления, используемых трансляторами с конкретных ЯП. Реализация интерфейсов между программами использует механизмы операционной системы (ОС). Обмен информацией происходит в основном на уровне файлов (включая файлы баз данных) и командных строк. Для вызова программы используются специальные средства супервизора ОС.

Выделение двух типов программных объектов позволяет установить связь сборочного программирования с методом модульного программирования и методами построения интегрированных комплексов. Таким образом, метод сборочного программирования является обобщением метода модульного программирования, и результаты, полученные при его анализе, будут справедливы для частных случаев.

2.4. ОСНОВНЫЕ МОДЕЛИ МЕТОДА СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

К основным моделям относятся модель информационного сопряжения и модель управления программными объектами.

Под **информационным сопряжением** двух или нескольких программных объектов понимается обеспечение преобразования множества их общих данных к представлениям, согласующимся с представлениями для каждого из объектов. В дальнейшем, не снижая общности анализа, рассматривается информационное сопряжение только пары объектов.

Модель информационного сопряжения представляет собой совокупность формальных описаний множеств данных для взаимодействующих объектов и функции их преобразования.

Под **управлением программными объектами** в рамках применения метода сборочного программирования понимается процесс планирования и выбора очередного объекта для выполнения и его активизация. При этом условиями выбора будут необходимость выполнения конкретной функции, следующей из алгоритма решаемой задачи, и

готовность входных данных. Модель управления программными объектами формализует описание условий для выбора объектов и определяет алгоритм самого выбора.

Указанные модели названы основными, так как проблемы передачи данных и управления являются главными задачами при объединении любых программных объектов с помощью подходящих методов и инструментальных средств.

2.4.1. МОДЕЛИ ИНФОРМАЦИОННОГО СОПРЯЖЕНИЯ

Рассмотрим процесс сборки объектов в интегрированный комплекс (ИК), определяемый как любой программный объект, состоящий из объединяемых компонент.

Пусть $P = \{p^i\}_{i=\overline{1,s}}$ —множество программных компонентов, входящих в состав создаваемого ИК. С каждой p^i связано множество данных D^i, посредством которого осуществляется информационный обмен между рассматриваемой и другими компонентами.

Множество $D^i = \{d^i_j\}_{j=\overline{1,t^i}}$ состоит из переменных d^i_j, каждая из которых характеризуется тройкой: именем (идентификатором переменной) N^i_j; типом T^i_j и текущим значением V^i_j.

Рассмотрим две программные компоненты p^i и p^k (p^k выполняется после p^i) со множествами данных D^i и D^k соответственно. В общем случае в D^i и D^k могут входить переменные, общие для p^i и p^k с точки зрения их семантической обработки. Эти переменные образуют подмножества $\tilde{D}^i$ и $\tilde{D}^k$. Задача информационного сопряжения состоит в преобразовании подмножества данных $\tilde{D}^i$ в представление, согласующееся с $\tilde{D}^k$.

Введем обозначения $N^i = \{N^i_j\}_{j=\overline{1,t^i}}$, $T^i = \{T^i_j\}_{j=\overline{1,t^i}}$, $V^i = \{V^i_j\}_{j=\overline{1,t^i}}$. В $\tilde{D}^i$ им соответствуют множества $\tilde{N}^i$, $\tilde{T}^i$ и $\tilde{V}_i$. В общем случае для преобразования множества данных D^i необходимо построить преобразования для $\tilde{N}^i$, $\tilde{T}^i$, и $\tilde{V}^i$. Рассмотрим следующие два случая.

1. Каждой переменной $d^i_j \in \tilde{D}^i$ соответствует только одна переменная $d^k_l \in \tilde{D}^k$. Тогда преобразование (отображение)

$$F^{ik} : \tilde{D}^i \to \tilde{D}^k \tag{2.1}$$

состоит из множества преобразований для отдельных переменных: $F^{ik} = \{F^{ik}_{jl}\}$. При этом формально $F^{ik}_{jl} = (FN^{ik}_{jl}, FT^{ik}_{jl}, FV^{ik}_{jl})$. Вводя обозначения $FN^{ik} = \{FN^{ik}_{jl}\}$, $FT^{ik} = \{FT^{ik}_{jl}\}$, $FV^{ik} = \{FV^{ik}_{jl}\}$, мы определяем преобразования

$$\begin{aligned} FN^{ik} &: \tilde{N}^i \to \tilde{N}^k, \\ FT^{ik} &: \tilde{T}^i \to \tilde{T}^k, \\ FV^{ik} &: \tilde{V}^i \to \tilde{V}^k \end{aligned} \tag{2.2}$$

для множеств идентификаторов, типов данных и значений соответственно.

2. Между переменными d_j^{li} и d_l^k не существует однозначного соответствия. Это имеет место, когда несколько элементов из $\tilde{D}^i$ соответствуют одному элементу из $\tilde{D}^k$ и наоборот. Сложная связь, при которой несколько элементов из $\tilde{D}^i$ соответствуют нескольким элементам из $\tilde{D}^k$, в практике сборочного программирования, как правило, отсутствует, что связано с раздельной разработкой отдельных программных компонент.

Соответствие нескольких переменных одной и наоборот свидетельствует об изменении уровня структурирования данных. Пусть $\tilde{d}_j^i$ соответствует несколько элементов из $\tilde{D}^k$. Обозначим их через $\tilde{d}_{l_1}^k, \ldots, \tilde{d}_{l_r}^k$, через S — функцию селектора, уменьшающую уровень структурирования данных:

$$S(\tilde{d}_{j_1}^i) = (\tilde{d}_{j1}, \ldots, \tilde{d}_{jr}^l), \tag{2.3}$$

где каждому $\tilde{d}_{j_v}^l$ соответствует $\tilde{d}_{l_v}^k$ при $v = 1, 2, \ldots, r$. Заменяя $\tilde{d}_j^i$ в $\tilde{D}^i$ элементами $d_{j_1}^i, \ldots, d_{jr}^i$, мы получаем множество $\tilde{\tilde{D}}^i$. Построение отображения $F^{ik} : \tilde{\tilde{D}}^i \to \tilde{D}^k$ производится аналогично первому случаю.

При соответствии нескольких элементов из $\tilde{D}^i$ одному элементу из $\tilde{D}^k$ поступаем следующим образом. Вместо функции селектора вводим функцию конструирования вида

$$C(\tilde{d}_{j_1}^i, \ldots, d_{jr}^i) = \tilde{d}_j^l, \tag{2.4}$$

где $\tilde{d}_j^l$ соответствует единственному элементу из $\tilde{D}^k$. Модифицируя множество $\tilde{D}^i$ и рассматривая отображение F^{ik}: $\tilde{\tilde{D}}_l \to \tilde{D}^k$, приходим к аналогичному результату.

Проведем анализ отображений FN, FT и FV (индексы для простоты опущены). Из построения следует, что множества $\tilde{N}^i$ и $\tilde{N}^k$ содержат одинаковое количество элементов. Поэтому FN только переупорядочивает идентификаторы переменных в соответствии с последовательностью, принятой при описании программной компоненты p^k.

Отображение преобразования множества типов данных FT является наиболее сложным, что связано с наличием практически неограниченного количества типов. По определению тип данных характеризуется парой $T = (X, \Omega)$, где X — множество значений, которые могут принимать переменные рассматриваемого типа, а Ω — множество операций, выполняемых над этими переменными. В данном случае T может рассматриваться как алгебраическая система. Под преобразованием типа $T_j^i = (X_j^i, \Omega_j^i)$ в тип $T_l^k = (X_l^k, \Omega_l^k)$ понимается преоб-

разование множества X_j^i в X_l^k, при котором семантическое содержание операций из Ω_l^k эквивалентно операциям из Ω_l^k.

В общем случае преобразование T_j^i в T_l^k может быть односторонним. Однако для повторного использования данных, что характерно для многократного вызова определенных программных компонент, обрабатывающих одни и те же структуры данных, нужны и прямое и обратное преобразования. Для достижения этого необходимо, чтобы отображение между T_j^i и T_l^k было изоморфизмом. Поэтому построение преобразования между двумя типами данных будет соответствовать нахождению изоморфного отображения между двумя алгебраическими системами.

При практической реализации модель информационного сопряжения целесообразно рассматривать как совокупность моделей для пар программных компонент $M = \{M^{ik}\}$. Модель для каждой из них имеет вид

$$M^{ik} = (\tilde{N}^i, \tilde{T}^i, \tilde{V}^i, \tilde{N}^k, \tilde{T}^k, \tilde{V}^k, FN^{ik}, FT^{ik}, FV^{ik}). \qquad (2.5)$$

Отдельные составляющие данной модели и принципы их построения описаны выше.

2.4.2. МОДЕЛИ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ОБЪЕКТАМИ

Анализируя критерии выбора программных объектов, необходимо отметить два случая.

1. Более приоритетным является критерий выполнения необходимой функции согласно алгоритму решения задачи. В этом случае происходит целенаправленная подготовка входных данных для выбранного программного объекта. Такая ситуация характерна для большинства методов передачи управления. Примерами могут служить оператор вызова ЯП, где с именем вызываемого модуля указывается список передаваемых параметров, и пакет операторов языка управления заданиями ОС ЕС, содержащий последовательность вызываемых программ и описания необходимых файлов. Вызов объекта происходит из предположения, что необходимые входные данные уже подготовлены ранее.

2. Приоритетным является критерий готовности данных. Вызывается тот объект, для которого подготовлена входная информация. Данный случай встречается значительно реже, и его целесообразно рассмотреть на конкретном примере.

На рис. 2.1 представлена схема интегрированного комплекса, состоящего из четырех программ (P_1, P_2, P_3, P_4). P_1 создает файлы F_1 и F_3. F_1 служит входной информацией для программы P_2, результатом которой является файл P_2. Аналогично результат работы F_3 — файл F_4. Программа P_4 исполь-

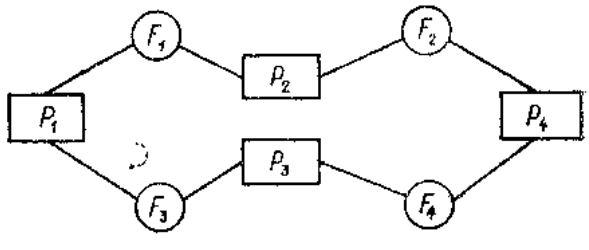


Рис. 2.1. Пример схемы интегрированного комплекса

зует файлы F_2 и F_4. Рассмотрим последовательность вызовов программ, основываясь на критерии готовности данных.

Первой будет выполняться программа P_1, так как она единственная не требует входной информации. После ее выполнения может быть вызвана P_2 или P_3. Выберем программу P_2. После ее выполнения согласно критерию будет вызвана P_3, так как для P_4 требуется файл F_4. Последней выполняется программа P_4. Аналогичный результат будет, если программа P_3 выполнится раньше, чем P_2. Таким образом, критерию готовности данных может удовлетворять несколько программных объектов, и любой из них может быть выбран в качестве очередного.

Оба рассмотренных случая можно объединить в рамках единой модели. Для этого введем специальный тип данных — «программные» переменные (PROGVAR). Этим переменным могут присваиваться имена выполняемых объектов. Переменные типа PROGVAR объединяются с данными для программных объектов, и рассматриваемый критерий применяется к объединению. Перейдем к формальному описанию модели управления.

Пусть $P = \{p^i\}_{i=\overline{1,s}}$ — множество программных компонент и множество D^i соответствует p^i, как и для модели информационного сопряжения. Рассмотрим множество данных интегрированного комплекса, определяемое как

$$D = (\bigcup_{i=1}^{s} D^i) \cup D^c, \tag{2.6}$$

где D^c обозначает множество управляющих данных. С каждой компонентой p^i свяжем предусловие R^i и постусловие Q^i, заданные на множестве D. R^i проверяется перед вызовом p^i и определяет условие вызова данной компоненты. Q^i проверяется после вызова p^i и определяет условие завершения решаемой задачи или выбор следующей компоненты. Пусть $R = \{R^i\}_{i=\overline{1,s}}$ и $Q = \{Q^i\}_{i=\overline{1,s}}$. Суть модели управления программными объектами состоит в следующем. При выборе очередной компоненты просматривается множество Q до первого истинного условия. Для данного условия проверяется соответствующее условие из R. Если последнее истинно или может быть к нему приведено (например, вводом необходимой информации с терминала пользователем), то происходит вызов соответствующей компоненты. Если условие из R ложно, то происходит поиск следующего истинного условия в Q и т. д. Последовательность вызовов компонент в общем случае не детерминирована, что обеспечивает динамические связи между компонентами.

Множество управляющих данных D^c содержит переменные, входящие в условия и не связанные со множествами D^i. В частности, к ним относятся переменные:

содержащие имена программных компонент интегрированного комплекса;

определяющие характеристики среды выполнения (технические условия, версия операционной системы и т. д.);

пользователя.

Наличие «программных» переменных вводит элементы детерминизма путем упорядочивания вызовов программных компонент. Подробный анализ рассматриваемой модели показывает, что задача выбора очередной программной компоненты аналогична задаче логического программирования. При этом условия R^i и Q^i соответствуют правилам для логического вывода, содержащим предикаты, функции и переменные из множества D. Множество предикатов и функций не фиксировано и может содержать операции отношения, арифметические операции, операции над файлами и т. д. В состав данного множества также входят специальные функции, позволяющие проверять готовность данных, обращение к пользователю для ввода информации с терминала или выбора одной из множества компонент, для которых выполняется истинность условий, и т. д. С точки зрения логического программирования условия R^i и Q^i равноценны, а порядок их проверки определяется выбранной стратегией обработки.

Опишем средствами логического программирования модель управления для ИК, схема которого приведена на рис. 2.1 (в нем для простоты верхний индекс переменной P заменен нижним). Введем следующие предикаты и функции:

select (x) — выбор компоненты с именем x;
def (y) — анализ готовности данного y;
undex (y) — анализ отсутствия готовности данного y;
exec (x) — выполнение компоненты с именем x;
setdef (y) — установка признака готовности для данного y;
cleardef (y) — сброс признака готовности для данного y.

Рассмотрим следующее описание:

1: select (P_4) : — def (F_2), def (F_4), exec (P_4), cleardef (F_2), cleardef (F_4)
2: select (P_3) : — def (F_3), exec (P_3), setdef (F_4), cleardef (F_3)
3: select (P_2) : — def (F_1), exec (P_2), setdef (F_2), cleardef (F_1)
4: select (P_1) : — undef (F_1), undef (F_3), exec (P_1), setdef (F_1), setdef (F_3).

Запрос на выбор очередной компоненты имеет вид ? select (x). Первому запросу будет удовлетворять правило 4. Его выполнение связано с вызовом программы P_1 и установкой признаков готовности для файлов F_1 и F_3. Второму запросу ? select (x) удовлетворяют правила 2 и 3, третьему — правило 4. После этого все признаки приведены в начальное состояние, и четвертому запросу будет удовлетворять правило 4.

Выбор и выполнение очередной компоненты можно автоматизировать. Для этого приведенная логическая программа будет иметь следующий вид:

1: select (x) : — $S(x)$
2: $S(P_4)$: — def (F_2), def (F_4), exec (P_4), cleardef (F_2), cleardef (F_4)

3: $S(P_3)$: — def (F_3), exec (P_3), setdef (F_4), cleardef (F_4), $S(x)$

4: $S(P_2)$: — def (F_1), exec (P_2), setdef (F_2), cleardef (F_1), $S(x)$

5: $S(P_1)$: — undef (F_1), undef (F_3), exec (P_1), setdef (F_1), setdef (F_3), $S(x)$.

Программа выполняет аналогичные действия, но смена компонент происходит автоматически. Рассмотренные логические программы построены на основе второго критерия — критерия готовности данных. Приведем два варианта программы, основанные на первом критерии:

1: select (x) : — $S(P_4)$

2: $S(P_4)$: — $S(P_3)$, exec (P_4)

3: $S(P_3)$: — $S(P_2)$, exec (P_3)

4: $S(P_2)$: — $S(P_1)$, exec (P_2)

5: $S(P_1)$: — exec (P_1).

Согласно этому описанию происходит последовательное выполнение компонент P_1, P_2, P_3 и P_4. Во втором варианте используется «программная» переменная. Для операций над ней введены предикат eq (Y, P), проверяющий равенство значения переменной Y имени переменной P, и функция set (Y, P), присваивающая переменной Y имя P:

1: select (x) : — set (Y, P_1), $S(Y)$

2: $S(Y)$: — eq (Y, P_4), exec (P_4)

3: $S(Y)$: — eq (Y, P_3), exec (P_3), set (Y, P_4), $S(Y)$

4: $S(Y)$: — eq (Y, P_2), exec (P_2), set (Y, P_3), $S(Y)$

5: $S(Y)$: — eq (Y, P_1), exec (P_1), set (Y, P_2), $S(Y)$

Практическая реализация описанного механизма выбора и вызова компоненты может выражаться в виде программы, написанной на языке ПРОЛОГ или на другом ЯП высокого уровня. Окончательно модель управления интегрированным комплексом примет следующий вид:

$$M = (P, D, R, Q, I), \tag{2.7}$$

где P — множество программных компонент, имена которых включены в пред- : и постусловия; D — множество данных, определяемое (2.6); $R = R(D)$ — множество предусловий, определенных на D; $Q = Q(D)$ — множество постусловий, определенных на D; I — средства, определяющие стратегию обработки правил (в практической реализации — это средства транслятора или интерпретатора).

Приведенная модель может использоваться для определения параллельно выполняемых программных объектов. Программы P_2 и P_3 (см. рис. 2.1) могут выполняться параллельно (при наличии соответствующих вычислительных ресурсов). Этот факт подтверждается тем, что второму запросу ? select (x) удовлетворяют два правила. Таким образом, средствами логического программирования могут быть описаны условия параллельного выполнения программных компонент.

Глава 3

КОМПЛЕКСИРОВАНИЕ МОДУЛЕЙ

Модульный принцип является основополагающим в процессе проектирования и разработки программных средств различного назначения. Он обеспечивает декомпозицию исходной задачи на ряд функций, каждая из которых реализуется программным модулем, осуществляющим преобразование исходных данных функций в выходные результаты. Модуль представляет собой самостоятельную, элементарную единицу процесса конструирования по методу сборочного программирования.

При сборке ПС из большого числа модулей с различными характеристиками и свойствами возникает проблема интерфейса модулей, состоящая в организации передачи данных (их импортирование и экспортирование) и управлений между объединяемыми модулями.

3.1. ОПРЕДЕЛЕНИЕ МОДУЛЯ И ЕГО СВОЙСТВ

Ранее было отмечено, что общепринятого формального определения понятия модуль в настоящее время не существует. В общем случае под **модулем** понимается преобразование множества исходных данных X во множество выходных данных Y, задаваемое в виде отображения

$$M : X \to Y. \qquad (3.1)$$

На сами множества X и Y, а также на отображение M накладывается ряд ограничений и дополнительных условий, позволяющих отделить модуль как самостоятельный программный объект от других классов программных объектов. Теперь будем говорить о том, что модуль обладает множеством свойств и характеристик. Рассмотрим эти свойства относительно трех основных этапов жизненного цикла ПС: проектирования, разработки и выполнения.

Этап проектирования структуры ПС определяет следующие свойства модулей:

необходимость выполнения одной или нескольких взаимосвязанных функций [61, 133, 205];

логическая законченность и обособленность относительно этапа проектирования (модуль должен обеспечивать выполнение необходимой функции независимо от результатов проектирования остальных компонент ПС) [111, 133, 144, 210];

независимость одного модуля от других (внутренняя логика данного модуля явно не связана с логикой работы других) [111, 167, 186, 194];

замена отдельного модуля в ПС без нарушения всей структуры [194, 205, 208];

возможность вызова других модулей и возврат управления вызвавшему модулю [76, 210];

уникальность именования модуля [76];

доступ к общим данным [111, 133, 144].

На этапе разработки отдельных модулей для них характерны следующие свойства:

раздельная компиляция [76];

описание на одном из языков программирования и представление в виде процедуры, подпрограммы, программной секции [61, 76];

один вход в модуль [76];

информация о характеристиках модуля должна содержаться в его описании [144,210];

ограничения на размер модуля [70, 76, 111, 133, 210].

Этап выполнения ПС предъявляет следующие требования к свойствам модуля:

возможность использования в различных местах программной системы [144, 205, 210];

отсутствие необходимости сохранять историю своих вызовов и обеспечение повторной входимости [76, 111];

передача данных между модулями через параметры вызова [111, 144];

изменение как можно меньшего количества передаваемых модулю данных [111];

унификация механизмов передачи управления и данных между модулями [119].

Не все свойства одинаково важны в процессе разработки ПС. Некоторые из них, как, например, ограничение на размер модуля, имеют различные оценки. Однако для модульного процесса проектирования характерны две обобщенные тенденции — усиление внутренних связей в модуле и ослабление внешних [61, 111]. Хорошо спроектированный модуль обладает значительно более сильными внутренними связями, чем внешними. При выполнении этого условия в процессе комплексирования модули можно рассматривать как неделимые программные единицы с определенными свойствами без учёта их внутренней структуры. Исходя из этого выделим свойства модулей, наиболее важные для процесса комплексирования.

Свойство 1. Логическая законченность и обособленность относительно этапа проектирования, позволяющая представить процесс комплексирования как композицию независимых функций.

Свойство 2. Заменяемость отдельного модуля в программной структуре, позволяющая рассматривать комплексирование как отдельный процесс разработки ПС, не связанный с существенными изменениями всей структуры.

Свойство 3. Свойство возврата управления непосредственно вызывающему модулю, что позволяет сводить процесс комплексирования к объединению пар модулей, взаимодействующих по передаче и возврату управления.

Свойство 4. Обращение одного модуля к другим, что позволяет строить программные структуры, состоящие из модулей,— модульные структуры.

Свойство 5. Уникальное именование модулей. Выполнение этого свойства гарантирует правильное построение и использование необходимых модулей в модульной структуре.

Свойство 6. Раздельная компиляция, при которой все модули в создаваемой программной структуре должны быть внешними. Данное условие позволяет отделить процесс трансляции отдельных модулей от процесса комплексирования.

Свойство 7. Наличие одного входа в модуль, что позволяет корректно решать задачи сопряжения каждой пары взаимодействующих модулей.

Свойство 8. Возможность использования модуля в различных местах ПС. Наличие этого свойства позволяет избегать дублирования объектов на этапе комплексирования, если к модулю имеется более одного обращения.

Свойство 9. Унификация механизмов передачи управления и данных между модулями. Выполнение этого условия обеспечивает возможность решения задач сопряжения отдельных модулей без учета их внутренней структуры, используя стандартизованные механизмы взаимодействия модулей.

Приведенные свойства позволяют выбрать аксиоматический подход к определению модуля.

Модулем *в рамках применения метода сборочного программирования называется программный объект, характеризующийся свойствами 1—9.*

Данное определение задает границы применения понятия модуль и справедливо для метода сборочного программирования. Операциями над объектами типа *модуль* являются различные функции комплексирования, характеристика которых будет приведена ниже.

3.2. ОПРЕДЕЛЕНИЯ И ХАРАКТЕРИСТИКИ МОДУЛЬНЫХ СВЯЗЕЙ

В данном разделе рассматриваются определения и характеристики взаимосвязей между модулями, используемые в дальнейшем для описания функций межмодульного интерфейса.

Под **видом связи** между модулями будем понимать: 1) связь по управлению; 2) информационную связь (связь по данным).

1. Связь по управлению характеризуется наличием или отсутствием среды ЯП и механизмом вызова.

Среда ЯП определяется как совокупность дополнительных программно-информационных средств окружения модулей, генерируемых компилятором с ЯП и включающих:

системные программные средства этапов выполнения, состоящие из библиотечных программ взаимодействия с операционной системой, вычисления стандартных функций, обработки внутренних структур данных и т. д.;

внутренние структуры данных, включающие различные системные данные, списки ассоциаций идентификаторов для программ блочной структуры, стек в виде последовательности точек передачи и возврата управления, и т. д.;

динамические данные, включающие списки областей используемой и свободной памяти, областей сохранения регистров и т. д.

Механизм вывоза определяется способом обращения к вызываемому модулю. Различают стандартный и нестандартный механизмы вызова. Стандартный определяется оператором вызова CALL ЯП, процедурным вызовом и вызовом функций средствами ЯП. Нестандартный представляет собой любой механизм, отличный от стандартного. Используется только при программировании на Ассемблере.

Связь по управлению обладает определенной сложностью, описываемой следующей функцией:

$$CP = K_1 + K_2, \tag{3.2}$$

где K_1 — коэффициент типа механизма вызова; K_2 — коэффициент перехода от среды ЯП вызывающего модуля к среде ЯП вызываемого.

Для стандартного механизма вызова $K_1 = 1$, для нестандартного — $K_1 = 1 + a$ $(a > 0)$. Здесь a зависит от количества отличных от стандартного характеристик вызова. К характеристикам вызова относятся: способ определения точки входа в вызываемый модуль; механизм передачи управления; формирование адреса возврата в вызывающий модуль; доступ к списку параметров; метод сохранения и восстановления регистров вызывающего модуля.

Коэффициент K_2 зависит от количества операций, необходимых для перехода от среды вызывающего модуля к среде вызываемого и наоборот. Аналитического выражения для K_2 не существует, но можно указать параметры, от которых он зависит. К ним относятся: количество библиотечных модулей, входящих в среду; количество выполняемых ими функций; количество структур данных, входящих в среду; общий объем памяти, занимаемой программно-информационными средствами среды; ЯП вызывающего и вызываемого модулей.

Если вызывающий и вызываемый модули написаны на одном ЯП, то $K_2 = 0$. Для остальных случаев $K_2 > 0$.

2. Информационная связь определяется как обмен данными между модулями. Различают регулярную и нерегулярную информационную связь. Регулярная характеризуется целенаправленным обменом постоянно определенного множества данных при каждой активизации вызываемого модуля. Этот тип связи реализуется посредством списка передаваемых параметров в операторах вызова. Нерегулярная — непостоянством множества обмениваемых данных или косвенным доступом (через посредников) к информации.

К этому типу связи относится обмен данными посредством глобальных имен или через внешние файлы.

Информационная связь характеризуется сложностью ПС, описываемой следующей функцией:

$$CI = \sum_{i=1}^{n} K_i F(x_i), \tag{3.3}$$

где K_i — весовой коэффициент для i-го параметра; $F(x_i)$ — функция количества элементов простых типов для параметра x_i.

Коэффициенты $K_i = 1$ — для простых переменных и $K_i > 1$ — для сложных типов данных. $F(x_i) = 1$, если x_i — простая переменная, и $F(x_i) > 1$ — для сложных типов данных. Необходимо отметить, что в (3.3) входят данные, принадлежащие к регулярной и нерегулярной информационной связи. Определение простых и сложных типов данных будет приведено ниже.

3.3. ВИДЫ ИНТЕРФЕЙСОВ И ИХ ФУНКЦИИ

В сборочном программировании важное место занимает интерфейс, под которым понимается взаимосвязь программных объектов (модулей, программ, языков и т. п.), обеспечивающая условия их совместного функционирования.

В зависимости от способа реализации интерфейсы могут быть встроенными, внешними и комбинированными [45]. Функции встроенного интерфейса реализуются в самих программных объектах. Внешний интерфейс разрабатывается как самостоятельный программный продукт. Функции комбинированного интерфейса частично реализуются в программных компонентах, а частично в виде отдельных программных средств.

В основе построения интерфейсов лежит:

стандартизация объектов, их свойств и характеристик;

разработка формализованных правил (механизмов) сопряжения стандартизованных объектов;

средства автоматизации механизмов связи объектов.

При этом имеет место несколько видов интерфейсов: межъязыковый, межмодульный, межпрограммный, пользовательский.

Межъязыковый интерфейс — это связь различных ЯП по типам содержащихся в них данных, методам их организации и способам отображения этих средств соответствующими системами программирования.

Основными функциями межъязыкового интерфейса является решение следующих четырех задач.

1. Обеспечение перехода от среды функционирования одного ЯП к среде функционирования другого. При связи разноязыковых модулей необходимо осуществить переход от среды ЯП вызывающего модуля к среде ЯП вызываемого. Перед возвратом управления необходимы обратные операции.

2. Обеспечение передачи управления между разноязыковыми модулями. Решение данной задачи связано с решением предыдущей. Если реализованы средства перехода от одной среды функционирования к другой, то передача управления между разноязыковыми модулями не отличается от передачи управления между одноязыковыми. В этом случае задача для межъязыкового интерфейса сводится к контролю за последовательностью обращения от вызывающего модуля к вызываемому.

Необходимо отметить, что решение данной задачи предполагает механизм непосредственного взаимодействия модулей без дополнительных средств обеспечения передачи управления (например, операционной системы).

3. Обеспечение доступа к общим данным. Механизмы доступа относятся к механизмам нерегулярной информационной связи, отличаются и зависят от места расположения информации.

Данные могут находиться в оперативной памяти, и тогда для доступа к ним используются такие средства языков программирования, как общие области для ЯП Фортран [15], данные типа EXTERNAL в ЯП ПЛ/1 [29], данные, описываемые операторами ENTRY, EXTRN, WXTRN в языке Ассемблер [129], и т. д. Если данные имеют файловую структуру, то они обычно хранятся во внешней памяти. В этом случае для доступа к ним используются операторы ввода—вывода ЯП высокого уровня и макрокоманды обмена языка Ассемблер.

Использование общих данных, расположенных в оперативной памяти, всегда сопряжено с опасностью их разрушения (намеренного или случайного характера) и с неверным выполнением программы. Поэтому при модульном подходе основным механизмом информационного обмена является аппарат передачи данных через параметры оператора вызова CALL ЯП высокого уровня и соответствующей макрокоманды языка Ассемблер. В случае особых режимов обработки общие данные могут передаваться через внешнюю память.

4. Реализация механизма передачи данных через параметры вызова. Этот механизм передачи данных относится к механизму регулярной информационной связи и наиболее стандартизован, так как он применяется практически во всех ЯП, допускающих секционирование программ. Задачей межъязыкового интерфейса в этом случае будет преобразование данных из списка фактических параметров вызывающего модуля к представлению, согласующемуся с описанием формальных параметров вызываемого модуля.

Преобразование направлено на устранение отличий как языковых, так и связанных с реализацией (представлением) данных системами программирования. Оно выполняется перед и после выполнения вызываемого модуля (т. е. осуществляется прямое и обратное преобразования).

Межмодульный интерфейс — это обеспечение связи модулей, записанных в разных ЯП, и управление модульными структурами программ.

Результат анализа свойств модуля для комплексирования позволяет сделать вывод о существовании двух больших групп задач для

межмодульного интерфейса. К первой относятся проблемы локального взаимодействия каждой пары модулей (свойства 3, 4, 7, 9), ко второй — проблемы построения и обработки модульных структур (свойства 1, 2, 5, 6, 8). Такое деление упорядочивает разработку межмодульного интерфейса, отделяя задачи сопряжения пар модулей от задач управления модульными структурами в целом. В соответствии с этим в задачу реализации межмодульного интерфейса входят разработка межъязыкового интерфейса как программной компоненты, автоматизирующей сопряжение пар модулей, и управление построением и обработкой модульных структур программ.

С о п р я ж е н и е пар разноязыковых модулей сводится к устранению отличий в:

1) языковых средствах ЯП;

2) описании отдельных модулей;

3) способах представления модулей системами программирования с ЯП.

1. Отличия в языковых средствах ЯП являются следствием неодинаковости синтаксического и семантического представления типов данных ЯП, их функциональных возможностей. К ним необходимо отнести:

механизм конструирования новых типов данных отсутствует в языках Фортран, ПЛ/1, Кобол и имеется в языках Паскаль, Ада, Симула-67, Модула-2, Си, Альфард, CLU;

некоторые предопределенные типы отсутствуют в определенных ЯП (например, символьный тип в ЯП Фортран);

представление некоторых предопределенных типов отличается в разных ЯП (логический тип в языке ПЛ/1 представлен как битовая строка);

динамические типы данных отсутствуют в Фортране и Коболе и имеются в языках Паскаль, ПЛ/1, Ада, Симула-67 и др.;

организация внешних файлов различна (ПЛ/1 допускает последовательную, индексно-последовательную и прямую организацию файлов, Фортран не обеспечивает индексно-последовательной организации файлов);

дескрипторы для представления структурных типов данных имеются в некоторых ЯП и не требуются в Фортране и Коболе;

представление некоторых структурных типов отличается в различных ЯП (массивы в Фортране располагаются по столбцам, в других ЯП — по строкам).

2. Проблемы сопряжения, связанные с описаниями модулей, вызваны несоответствием задания формальных и фактических параметров и состоят в следующем:

описания типов данных, областей значений переменных, индексов массивов и т. д. задаются неоднозначно;

с одним формальным параметром сопоставляется несколько фактических и наоборот, что порождается отсутствием структурных типов данных в некоторых ЯП и их обработкой в виде нескольких отдельных компонент;

изменение порядка следования параметров.

3. К проблемам, связанным с реализацией систем программирования, относятся:

особенности передачи управления (наличие среды функционирования);

различия во внутреннем представлении однородных типов данных для различных систем программирования;

различия в структуре и организации внешней памяти для однородных файлов.

Из приведенного выше следует, что проблема передачи управления между разноязыковыми модулями носит не принципиальный характер, а является следствием реализации конкретных систем программирования с ЯП. Это подтверждается также тем, что аналогичные проблемы для ЯП, реализованных на мини- и микроЭВМ, практически отсутствуют или незначительны.

Управление построением и обработкой модульных структур программ как одной из задач межпрограммного интерфейса основывается на реализации следующих четырех функций.

1. Комплексирование программных средств различного уровня сложности. Этот процесс является довольно ответственным этапом в разработке ПС и выполняется обычно в несколько шагов. На каждом приходится иметь дело с программными агрегатами различного уровня сложности, готовности, прошедшим различные стадии отладки и т. д. Комплексирование таких разнородных программных компонент должно происходить на единой методологической основе, составляющей содержание данной задачи межпрограммного интерфейса.

2. Обеспечение построения различных видов программных структур. Существует несколько видов программных структур, каждая из них характеризуется особенностями построения и выполнения программного агрегата. Наибольшее распространение в практике программирования получили простая, оверлейная и динамическая структуры. Построение этих программных структур и составляет вторую задачу управления модульными структурами.

Выполнение операций над модульными структурами. Перед этапом комплексирования необходимо исследовать модульные структуры объектов для обеспечения правильности их функционирования. В частности, эти исследования состоят в выполнении операций определения доступности модулей и их именования, анализа циклов в последовательностях обращений между модулями и др. Поэтому в управлении обработкой модульных структур заключается третья задача межпрограммного интерфейса.

4. Обеспечение тестирования и отладки межмодульных переходов. Данная задача является комплексной, так как при ее решении используются как средства межъязыкового интерфейса, так и средства управления модульными структурами. Средства сопряжения модулей позволяют обеспечить тестирование передаваемых параметров, а средства управления модульными структурами — отслеживание цепочек выполняемых модулей. Решение данной задачи способствует построению правильных модульных структур.

3.4. ОСНОВНЫЕ МОДЕЛИ КОМПЛЕКСИРОВАНИЯ МОДУЛЕЙ

Во второй главе приведены основные модели сборочного программирования — модель информационного сопряжения и модель управления программными объектами. Комплексирование модулей как частная проблема сборочного программирования также может, в общем, описываться этими моделями. Однако конкретное определение объектов — модулей — позволяет детализировать данные модели. Детализация основывается на следующих основных факторах.

1. Использование оператора вызова CALL или ему подобного для обращения к модулю. Этот способ соответствует первому случаю выбора программных объектов в модели управления (см. п. 2.4.2).

2. Обмен данными через параметры оператора вызова как основной способ информационного сопряжения. Передаваемые через параметры данные можно разделить на два типа — входные и выходные. Входные формируются в вызывающем модуле и используются в вызываемом без изменения их значений. Выходные данные формируются в результате работы вызываемого модуля и возвращаются вызывающему.

3. Описание средствами одного из ЯП высокого уровня или Ассемблера всех передаваемых данных. Это также относится к описаниям данных с различным уровнем структурирования (см. п. 2.4.1).

Эти факторы позволяют формализовать описание проблемы комплексирования модулей и построить алгоритм ее решения. Решение проблемы комплексирования модулей заключается в нахождении формализованного метода построения программных интерфейсов и управления модульными структурами.

Рассмотрим формальное описание задачи разработки межъязыкового интерфейса (МЯИ). Зафиксируем класс языков программирования L, состоящий из η ЯП: $L = \{l_\alpha\}$, $\alpha = 1, \ldots, \eta$. Пусть имеется пара взаимодействующих модулей, из которых вызывающий написан на l_α-языке, а вызываемый — на l_β-языке. Обозначим через $V = \{v^1, v^2, \ldots, v^k\}$ список фактических параметров вызывающего модуля, а через $F = \{f_1, f_2, \ldots, f^{k_1}\}$ — список формальных параметров вызываемого модуля. В общем случае $k \neq k_1$.

Разделим V и F в соответствии со множествами входных и выходных параметров: $V = V_i \cup V_0$, $F = F_i \cup F_0$ (i соответствует множеству входных, а 0 — множеству выходных параметров). При этом $V_i \sim F_i$, а $V_0 \sim F_0$ ($\sim$ обозначает знак соответствия).

Исходя из этих обозначений задача МЯИ заключается в преобразовании типов данных из V_i в соответствующее представление в F_i и F_0 — в представление V_0. При этом принципиальной разницы между преобразованиями типов данных от V_i к F_i и от F_0 к V_0 не существует (в дальнейшем индексы i и 0 при соответствующих множествах будем опускать).

В общем случае, как отмечено в п. 2.4.1, элементу множества F может соответствовать несколько элементов множества V и наоборот, что объясняется различием в типах данных для фактических и формальных параметров. Поэтому отображение для отдельных элементов множеств $V = \{v^1, v^2, \ldots, v^k\}$ и $F = \{f^1, f^2, \ldots, f^{k_1}\}$ может оказаться

неоднозначным. Для построения однозначного отображения необходимо провести разбиение множеств таким образом, чтобы каждому подмножеству из V соответствовало только одно подмножество из F. Проведем построение этого разбиения. Для каждого $f^i \in F$ рассмотрим его полный прообраз $V^i \in V$. Различные прообразы V^i и V^j могут иметь или не иметь одинаковые элементы. Если $V^i \cap V^j \neq \emptyset$, то объединим V^i и V^j в одно подмножество. Соответственно будет проведено объединение в одно подмножество элементов F^i и F^j. Данная процедура применяется до тех пор, пока не будет исчерпано множество V. В результате получим два семейства подмножеств $\Pi = \{V^1, V^2, \ldots, V^m\}$ и $\Phi = \{F^1, F^2, \ldots, F^m\}$ таких, что

$$\bigcup_{t=1}^{m} V^t = V, \; V^t \cap V^{t_1} = \emptyset \text{ при } t \neq t_1, \tag{3.4}$$

$$\bigcup_{t=1}^{m} F^t = F, \; F^t \cap F^{t_1} = \emptyset \text{ при } t \neq t_1, \tag{3.5}$$

для которых существует однозначное отображение, записываемое в виде

$$A : \Pi \to \Phi. \tag{3.6}$$

В зависимости от количества элементов во множествах V^t и F^t имеют место следующие случаи.

1) $|V^t| = |F^t| = 1$. Отображение А для данных подмножеств включает операции преобразования типов данных.

2) $|F^t| > 1$ и $|V^t| = 1$. Это означает, что одному фактическому параметру структурного типа данных соответствует несколько формальных параметров скалярных типов или структурных с меньшим уровнем структурирования. Отображение А включает операции селектора отдельных компонент и преобразования типов данных.

3) $|V^t| > 1$ и $|F^t| = 1$. Это означает соответствие нескольких фактических параметров одному формальному. Отображение А содержит операции преобразования типов и конструирование структурного типа с более высоким уровнем структурирования, чем у передаваемых параметров.

4) $|V^t| > 1$ и $|F^t| > 1$. Это свидетельствует о существовании глубокой связи вызывающего и вызываемого модулей, которая зависит от внутренней логики функционирования модулей. Такие связи противоречат свойствам модулей, и поэтому данный случай не поддается формальному анализу при комплексировании модулей.

На основе проведенного анализа свойств отображения А выделим три класса операций для информационного сопряжения модулей.

1. **Операции преобразования типов данных (Р).** Позволяют осуществлять непосредственное преобразование типа данных T_α^t в T_β^q без дополнительных операций, изменения уровня структурирования. Операцию преобразования запишем в виде

$$P_{\alpha\beta}^{tq} = (P\, T_\alpha^t,\; T_\beta^q).$$

Здесь данные типа T^t_α преобразуются в T^q_β (α и β соответствуют языкам l_α и l_β). Предполагается, что множество типов данных каждого ЯП упорядочено и индексы t и q определяют конкретные элементы этого множества. Для ЯП, имеющих средства конструирования новых типов, t и q будут функциями от других индексов, и упорядоченность типов может определяться тем, что новый тип t будет конструироваться из типов, для которых индексы не больше t.

Каждый ЯП имеет определенное множество предопределенных типов данных и базовых операций конструирования, что определяет основу всего множества типов. Новый тип будет иметь индекс, функционально зависящий от индексов предопределенных типов и конкретных операций конструирования.

2. **Операции селектора (S).** Используются для выбора из структурного типа его отдельных компонент с меньшим уровнем структурирования. Механизмы реализации этих операций отличны от аналогичных, имеющихся в ЯП, так как они не должны изменять структуры данных, непосредственно обрабатываемых в модулях (выполняются свойства модулей, связанные с неизменностью их внутренней структуры при комплексировании).

3. **Операции конструирования структурных типов (C).** Данные операции являются обратными по отношению к операциям селектора. Их механизмы конструирования отличны от аналогичных операций, имеющихся в ЯП.

Множество рассматриваемых операций охватывает как преобразования типов данных для ЯП из класса L, так и необходимые функции конструирования структурных типов и выбора их отдельных компонент. Детальное рассмотрение показывает, что для данного множества операций полнота отсутствует. Причины этого рассматриваются ниже.

Исходя из приведенных определений формализуем постановку задачи создания МЯИ.

Дано: класс ЯП $L = \{l_1, l_2, \ldots, l_\eta\}$ и для каждого из ЯП известны множества типов данных и операций конструирования новых типов.

Необходимо:

1. Построить множества операций преобразования типов данных $P = \{P^{tq}_{\alpha\beta}\}$, операций селектора S и конструирования C для структурных типов.

2. Для каждой пары взаимодействующих модулей провести разбиения множеств фактических и формальных параметров в соответствии с формулами (3.4) и (3.5) и построить отображение A на основе P, S и C.

Если отображение A построить не удается, то это означает, что МЯИ не обеспечивает сопряжение данной пары модулей с соблюдением свойств модулей. Возможно, сопряжение может быть реализовано с нарушением рассматриваемых свойств. Последнее замечание позволяет использовать МЯИ для определения качества создаваемой модульной структуры.

Определим аппарат описания типов данных ЯП. Каждый тип данных характеризуется множеством значений, которые могут принимать

переменные этого типа, и множеством операций над этими переменными. Поэтому наиболее подходящим методом, как было отмечено в гл. 2, является описание типов данных как алгебраических систем.

Введем обозначение для алгебраической системы, соответствующей некоторому типу данных: $\mathfrak{U} = \langle X, \Omega \rangle$. Здесь X — множество значений рассматриваемого типа, Ω — множество операций над объектами данного типа.

Тип самой алгебраической системы определяется в соответствии со структурой множества Ω. В общем случае результаты некоторых операций из Ω могут не принадлежать X, и тогда $\mathfrak{U}$ может рассматриваться как частичная алгебраическая система.

Выбор данного метода описания обусловлен содержанием задачи информационного сопряжения для МЯИ. Операции преобразования типов $P^{tq}_{\alpha\beta}$ должны обеспечивать не только однозначное соответствие множеств значений преобразуемых типов данных в вызывающем и вызываемом модулях, но и одинаковую интерпретацию операций над данными в этих модулях. При этом должно осуществляться как прямое преобразование данных от вызывающего к вызываемому модулю, так и обратное.

При таком подходе операции преобразования $P^{tq}_{\alpha\beta}$ соответствуют изоформным отображениям одной алгебраической системы в другую. Отображение A, рассмотренное ранее, есть набор изоморфизмов, включающий изоморфное отображение множеств фактических и формальных параметров и изоморфные отображения алгебраических систем для типов передаваемых параметров. Исходя из этого первую часть постановки задачи формулируем в следующем виде.

Для заданного множества алгебраических систем $\sum = \{\mathfrak{U}^t_\alpha\}$ ($\mathfrak{U}^t_\alpha$ соответствует типу данных t в α-м ЯП из рассматриваемого класса L языков) построить все возможные изоморфные отображения между элементами множества $\sum$.

Таким образом, задача построения МЯИ будет заключаться в выполнении следующих действий.

1. Определение класса ЯП и построение множества $\sum = \{\mathfrak{U}^t_\alpha\}$ при условии, что типы данных T^t_α описываются как алгебраические системы $\mathfrak{U}^t_\alpha$,

2. Нахождение изоморфных отображений между элементами множества $\sum$, результатом которого для $\mathfrak{U}^t_\alpha$ в $\mathfrak{U}^q_\beta$ могут быть:

явный вид изоморфизма;

множество изоморфных отображений с определенными общими свойствами и ограничениями;

доказательство отсутствия существования изоморфного соответствия между анализируемыми алгебраическими системами.

Если имеет место последний результат, то полученное множество операций $P = \{P^{tq}_{\alpha\beta}\}$, соответствующее множеству изоморфных отображений, будет характеризоваться неполнотой, как было отмечено выше.

3. Для всех структурных типов данных строятся множества операций селектора S и конструирования C. Эти операции базируются на общем подходе к структурной организации данных без учета их особенностей в конкретных ЯП. Необходимость построения множеств S и C определяется задачами и возможностями конкретного МЯИ, и в частном случае эти множества могут быть пустыми.

4. Для каждой пары взаимодействующих модулей должно быть выполнено:

построение изоморфного отображения между множествами фактических V и формальных F параметров (построение множеств П и Ф);

выбор необходимых операций селектора и конструирования из множеств S и C;

выбор необходимых изоморфных преобразований из множества P.

Предложенная схема построения МЯИ базируется на типах данных ЯП, которые рассматриваются ниже.

3.5. ОСНОВНЫЕ ТИПЫ ДАННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Для анализа основных типов используется теория структурной организации данных [27, 151]. Она базируется на формализованном подходе к определению типов, основанном на аксиоматизации каждого типа и правилах выполнения операций над объектами. Система аксиом определяет структуру множества значений типа, принадлежность ему отдельных элементов, их основные свойства, связь с другими типами данных. Для каждой операции, выполняемой над переменными рассматриваемого типа, определяются типы операндов и результата.

Теория структурной организации данных не ориентирована на применение в конкретном ЯП. Среди существующих ЯП основные положения данной теории воплощены в Паскале [48, 71], Модула-2 [29], Ада [26] и др. В дальнейшем для обозначения типов данных используются синтаксические конструкции языка Паскаль.

Существуют четыре предопределенных типа данных: целый (INTEGER); вещественный (REAL); булевый (BOOLEAN); символьный (CHARACTER). Они характеризуются тем, что на уровне архитектуры большинства ЭВМ имеются средства для обработки соответствующих типов данных и они существуют практически во всех ЯП. Остальные являются производными и образуются с помощью средств конструирования новых типов данных.

Все типы данных делятся на простые и структурные. К простым относятся перечислимые и числовые, к структурным — массивы, записи, множества, списки, последовательности и т. д. В настоящей работе подробно рассматриваются типы данных: перечислимые на основе булевого и символьного типов; числовые на основе целого и вещественного типов; массивы и записи как объекты структурных типов.

Другие типы, представленные в теории структурной организации данных, являются менее распространенными в ЯП и рассматриваются без подробного анализа.

3.5.1. ПРОСТЫЕ ТИПЫ ДАННЫХ

К простым типам относятся перечислимые и числовые. Общее обозначение перечислимого типа имеет вид type $T = (x_1, x_2, \ldots, x_n)$. Здесь T — имя типа, а $x_1, x_2, \ldots, x_n$ — имена всех значений типа T, образующих множество значений X. Операции над перечислимыми типами включают бинарные операции отношения и унарные операции pred и succ, определяющие соответственно предыдущий и последующий элементы во множестве X. Все операции отношения ($<$, $\leqslant$, $>$, $\geqslant$, $=$, $\neq$) при построении алгебраических систем будут заменены одной ($\leqslant$), определяющей линейную упорядоченность множества X.

Примерами перечислимых типов могут служить булевый и символьный типы. Множество значений булевого типа состоит из двух элементов — false и true. Множество операций Ω, кроме перечисленных выше, включают операции булевой алгебры &, $\vee$, $\neg$ Запишем алгебраическую систему, соответствующую булевому типу:

$$\mathfrak{U}^b = \langle X^b, \Omega^b \rangle,$$
$$X^b = \{\text{false, true}\}, \qquad (3.7)$$
$$\Omega = \{\&, \vee, \neg, \text{pred, succ}, \leqslant\},$$

$\mathfrak{U}^b$ имеет тип $\langle 2, 2, 1, 1, 1;2 \rangle$ согласно арности соответствующих операций и предикатов.

Булевый тип в ЯП записывается в виде

$$\text{type } T^b = (\text{false, true}).$$

Если он стандартный (BOOLEAN), то это описание в текстах модулей может опускаться.

Множество значений X символьного типа состоит из букв, цифр и специальных символов (знаков арифметических операций, знаков препинания и т. д.). Множество операций совпадает со множеством операций для любого перечисленного типа. Алгебраическая система $\mathfrak{U}^c$, соответствующая символьному типу, имеет вид

$$\mathfrak{U}^c = \langle X^c, \Omega^c \rangle,$$
$$X^C = \{\ldots, \text{'}A\text{'}, \ldots, \text{'}X\text{'}, \ldots, \text{'0'}, \text{'1'}, \ldots, \text{'9'}\}, \qquad (3.8)$$
$$\Omega^C = \{\text{pred, succ}, \leqslant\}.$$

Множество X упорядочено согласно внутреннему представлению символов для ОС ЕС.

Алгебраическая система $\mathfrak{U}^c$ имеет тип $\langle 1,1;2 \rangle$ согласно арности соответствующих операций и предикатов.

Символьный тип средствами ЯП записывается следующим образом: type T^c = (..., 'A', ..., 'X', ..., '0', ..., '9'). Если он стандартный (CHAR), то это описание в модулях может быть опущено. Операция ord, присваивающая каждому символу его порядковый номер во множестве X^c, и chr, определяющая по порядковому номеру символа его значение, являются по существу операциями преобразования типов и поэтому во множество Ω^c не включены.

Для перечисленных типов характерны следующие аксиомы [151], которые в дальнейшем будут использоваться для анализа операций преобразования типов данных:

$$X.\min \in X,$$
$$X.\max \in X, \tag{3.9}$$
$$(\forall x \in X) \& (x \neq X.\max) \Rightarrow \operatorname{succ}(x) \in X,$$
$$(\forall x \in X) \& (x \neq X.\max) \Rightarrow \operatorname{succ}(x) \neq X.\min.$$

Здесь $X.\min$ и $X\max$ обозначают соответственно минимальный и максимальный элементы множества X.

Практическое использование числовых типов всегда содержит ограничения, определяемые архитектурой ЭВМ (конечное значение количества разрядов слова памяти для представления чисел) или явным описанием в модулях (для ограничения диапазона значений отдельных элементов). Поэтому все числовые типы без нарушения общности анализа могут быть рассмотрены как отрезки. В общей форме отрезок записывается в виде $\text{type}\, T = (X.\min .. X.\max)$. Здесь $X.\min$ и $X.\max$ обозначает соответственно минимальный и максимальный элементы отрезка. Для любого $x \in X$ выполняется условие $x.\min \leqslant x \leqslant x.\max$. Для стандартных числовых типов (INTEGER и REAL) приведенное выше описание в модулях может быть опущено. В этом случае элементы $X.\min$ и $X.\max$ не определены и зависят от конкретной реализации транслятора с ЯП на конкретном типе ЭВМ.

Над переменными целого типа и типов, для которых целый тип является базовым, выполняются те же операции, что и в случае перечислимых типов. Кроме того, добавляются операции целочисленной арифметики: унарный минус, $+$, $-$, $\times$, div (целочисленное деление) и mod (получение остатка от деления). Алгебраическая система $\mathfrak{U}^i$, соответствующая целому типу, будет иметь вид

$$\mathfrak{U}^i = \langle X^i, \Omega^i \rangle,$$
$$X^i = \{X^i \cdot \min, X^i \cdot \max + 1, \ldots, X^i \cdot \max\}, \tag{3.10}$$
$$\Omega^i = \{+, \times, \operatorname{div}, -, \leqslant\}.$$

Во множестве Ω_i операция «—» соответствует унарному минусу. Остальные операции выражаются через операции, включенные в Ω. Алгебраическая система $\mathfrak{U}^i$ имеет тип $\langle 2,2,2,1;2\rangle$ согласно арности операций и предикатов. Форма записи целого и отрезков целого типа в ЯП имеет вид

$$\text{type}\, T^i = (X^i \cdot \min .. X^i \cdot \max).$$

Над переменными вещественного типа и типов, для которых вещественный тип является базовым, выполняются операции отношения и обычные арифметические операции для действительных чисел

(унарный минус, $+$, $-$, $\times$, $/$). Алгебраическую систему $\mathfrak{U}^r$, соответствующую вещественному типу, запишем следующим образом:

$$\mathfrak{U}^r = \langle X^r,\ \Omega^r \rangle,$$
$$X^r = \{x \mid X^r \cdot \min \leqslant x \leqslant X^r \cdot \max\}, \tag{3.11}$$
$$\Omega^r = \{+,\ \times,\ /,\ -,\ \leqslant\}.$$

Во множестве Ω^r операция «—» соответствует унарному минусу. Алгебраическая система Ω^r имеет тип $\langle 2,2,2,1;2\rangle$ согласно арности операций и предикатов.

Форма записи вещественного и отрезков вещественного типа в ЯП имеет вид

$$\text{type } T^r = (X^r \cdot \min \cdot\cdot X^r \cdot \max).$$

Необходимо сделать замечание относительно порядка выполнения операций над любыми типами:

все операнды приводятся к базовому типу;

операция выполняется, как над объектами базового типа;

для полученного результата выполняется обратный переход от базового к исходному типу.

Если результат принадлежит множеству значений данного типа, то операция выполняется верно. В противном случае результат операции не определен.

В дополнение к аксиомам (3.9) для числовых типов будут использоваться следующие:

$$(\forall x \in X) \Rightarrow T\,(T^0\,(x)) = x, \tag{3.12}$$
$$(\forall x_1 \in X)\,\&\,(\forall x_2 \in X) \Rightarrow (x_1 \leqslant x_2) \equiv (T^0\,(x_1) \leqslant T^0\,(x_2)).$$

Здесь T^0 обозначает базовый тип для типа T. Операции $T^0(x)$ и $T\,(x)$ определяют преобразование значения к соответствующему типу. В этих обозначениях выполнение арифметических операций для числовых типов будет определяться следующим образом ($\oplus$ — любая двухместная арифметическая операция):

$$(\forall x_1 \in X)\,\&\,(\forall x_2 \in X) \Rightarrow (x_1 \oplus x_2) \equiv T\,(T^0\,(x_1) \oplus T^0\,(x_2)). \tag{3.13}$$

Операции сравнения для числовых типов выполняются согласно второй аксиоме (3.12).

3.5.2. СТРУКТУРНЫЕ ТИПЫ ДАННЫХ

Отличительной особенностью структурных типов данных в сравнении с простыми является то, что они содержат несколько упорядоченных элементов, обработка которых проводится как над целыми объектами, так и на уровне отдельных элементов.

Структурные типы строятся из базовых типов и отличаются функциями конструирования и механизмами обработки. В качестве основных структурных типов в работе рассматриваются массивы и записи.

Массивы

Функция конструирования *массива* на основе базовых типов состоит в определении отображения из множества индексов на множество значений его элементов [151]:

$$M : I \to Y. \tag{3.14}$$

Здесь I — множество индексов, Y — множество значений элементов массива. В общем случае отображение M может не быть взаимно однозначным, если в различных элементах массива (элементы с разными индексами) содержатся одинаковые значения. Множество I является множеством значений перечислимого типа или отрезка целого типа. Элементы множества Y могут быть элементами любого типа, допускаемого в теории структурной организации данных.

Над массивами могут выполняться следующие операции:
отношение для упорядоченных массивов (определяется как совокупность операций отношения для всех элементов массивов);

сложение и вычитание однотипных массивов, т. е. массивов с одним и тем же множеством индексов (определяется как совокупность соответствующих операций над всеми элементами массивов с одинаковыми индексами);

умножение двухмерных массивов по правилам умножения матриц.

Операция умножения накладывает ограничения на область значений индексов массивов, связанных с правилами умножения матриц, и не является общей для всех типов массивов. Операции сложения и вычитания выполняются только для числовых массивов. Поэтому они не входят в состав множества общих операций над массивами. Алгебраическая система, соответствующая типу данных *массив*, имеет следующий вид:

$$\begin{gathered}\mathfrak{U}^a = \langle X^a, \Omega^a \rangle, \\ X^a = \{x \mid (\forall x_1 \in X^a) \,\&\, (\forall x_2 \in X^a) \Rightarrow I(x_1) = \\ = I(x_2)) \,\&\, (Y(x_1) \cup Y(x_2) \subset \overline{Y}(X^a))\}, \\ \Omega^a = \{\leqslant\}.\end{gathered} \tag{3.15}$$

Здесь $I(x)$ обозначает множество индексов для массива x, $Y(x)$ — множество значений элементов массива x, $\overline{Y}(X^a)$ определяет множество значений элементов для всех массивов, принадлежащих рассматриваемому типу. Второе выражение (3.15) обозначает, что к данному типу принадлежат только те массивы, у которых множества индексов совпадают, а множество значений их элементов принадлежат одному и тому же множеству, характеризующему рассматриваемый тип. В обозначениях (3.15) отображение (3.14) примет следующий вид (I — постоянно для всех x):

$$x : I \to Y(x), \quad Y(x) \subset \overline{Y}(x). \tag{3.16}$$

Множество Ω^a состоит только из одного предиката. Поэтому алгебраическая система $\mathfrak{U}^a$ фактически является алгебраической моделью типа $\langle 2 \rangle$

Тип данных *массив* в ЯП записывается в виде

$$\text{type } T^a = \text{array } T\,(I) \text{ of } T\,(\overline{Y}),$$

где T^a — тип данных *массив*; $T\,(I)$ — тип данных индексов массива; $T(\overline{Y})$ — тип данных для множества значений элементов массивов типа T^a.

Операции, определенные в (3.15), выполняются над массивами как над единым структурным значением. Кроме того, над элементами множеств I и Y могут выполняться операции, соответствующие их типам данных. Определим операцию селектора для элементов массива. Пусть $I' \subset I$. Через E обозначим вложение $I' \to I$. Отображение $E \cdot M : I' \to Y$ называется ограничением отображения M на I' и обозначается $M \mid I'$ [93]. Если I' состоит из одного элемента $I' = \{k\}$, то $M \mid \{k\}$ будет определять элемент множества Y, соответствующий индексу k. Заменяя M на x (согласно определению массива), получим обозначение операции селектора для элементов массива $x \mid \{k\}$. В языках программирования обычно элемент массива обозначается в виде $x\,[k]$.

Необходимо отметить, что в предыдущем анализе рассматривались одномерные массивы. Многомерные массивы определяются рекурсивно. Данные типа $T\,(Y)$ в описании массива T^a в свою очередь могут быть массивами и т. д. При этом подходе последовательность предложений

$$\text{type } T^a = \text{array } T\,(I^1) \text{ of } T(Y^1),$$
$$\text{type } T\,(Y^1) = \text{array } T(I^2) \text{ of } T\,(Y^2)$$

эквивалентна следующей записи:

$$\text{type } T^a = \text{array } T\,(I^1 \times I^2) \text{ of } T\,(Y^2).$$

В последнем предложении множество индексов массивов, принадлежащих типу T^a, представлено в виде прямого произведения множеств значений для типов $T\,(I')$ и $T\,(I^2)$. Операция селектора будет определяться в виде $\{\{x\} \mid \{i\}\} \mid \{j\}\}$, если $i \notin I^1$ и $j \notin I^2$. В ЯП элемент массива в этом случае будет обозначаться через $x[i]\,[j]$ или $x\,[i, j]$.

Запись

Структурный тип *запись*, как и *массив*, состоит из нескольких компонент, которые могут быть разнородными, т. е. принадлежать различным простым или структурным типам. Функция конструирования записей представляет конкатенацию отдельных компонент. Множество значений типа *запись* является прямым произведением множества значений ее компонент. Ко множеству операций, выполняемых над записями, относятся только операции отношения. При этом предполагается, что сравниваться могут только однотипные структуры (типы компонент сравниваемых записей и их порядок следования одинаковы) и для каждой компоненты записи выполняемая операция отношения интерпретируется как соответствующая операция для типа, соответствующего данной компоненте.

Пусть запись состоит из n компонент. Каждая m-я компонента ($m = 1, 2, \ldots, n$) имеет тип T^{v_m}, и ей соответствует алгебраическая система $\mathfrak{U}^{v_m} = \langle X^{v_m}, \Omega^{v_m} \rangle$. Индекс v_m соответствует одному из индексов для рассматриваемых в работе типов данных. Алгебраическая система для *записи* будет иметь вид

$$\mathfrak{U}^z = \langle X^z, \Omega^z \rangle,$$

$$X^z = \{x \mid (x = x^{v_1} \times \cdots \times x^{v_n}) \,\&\, (x^{v_1} \in X^{v_1}) \,\&\, \cdots \,\&\, (x^{v_n} \in X^{v_n})\}, \qquad (3.17)$$

$$\Omega^z = \{\leqslant\}.$$

Аналогично массиву алгебраическая система $\mathfrak{U}^z$ является моделью типа $\langle 2 \rangle$.

Общая форма представления типа для записи имеет вид

$$\text{type } T^z = (S_{v_1} : T^{v_1}; \ \ldots S_{v_n} : T^{v_n}).$$

Здесь $S_{v_1}, \ldots, S_{v_n}$ является селекторами, а $T^{v_1}, \ldots, T^{v_n}$ — типами данных для компонентов записи. В ЯП запись описывается следующим образом ($S_{v_1}, \ldots, S_{v_n}$ — имена компонентов записи):

$$\begin{aligned} \text{type } T^z = \ & \text{record} \\ & S_{v_1} : T^{v_1} \\ & \vdots \\ & S_{v_n} : T^{v_n} \\ & \text{end}. \end{aligned}$$

Операции, введенные в (3.17), выполняются над записью как над единым структурным значением. Для обработки отдельных компонент введем операцию селектора, аналогично соответствующей операции для массива. Пусть $I = \{S_{v_1}, \ldots, S_{v_n}\}$, $I' \subset I$. Обозначим через E вложение $I' \to I$. Пусть x — переменная типа *запись* и определяется согласно (3.17). Введем в рассмотрение множество $X^v = \{x^{v_1}, \ldots, x^{v_n}\}$. Тогда между I и X^v существует однозначное соответствие $M : I \to X^v$. Ограничение отображения M на I' обозначим через $M \mid \{S_{v_m}\}$. Если I состоит из одного элемента $I' = \{S_{v_m}\}$, то $M \mid \{S_{v_m}\}$ будет определять S_{v_m}-ю компоненту записи. Заменяя M на x, получаем $x \mid \{S_{v_m}\}$. В ЯП компоненты записи обозначаются в виде $x \cdot S_{v_m}$, где S_{v_m} — имя соответствующей компоненты.

Проведенный анализ относится к фиксированным записям. Для вариантных записей, последовательность компонент которых определяется специальным признаком, можно поступить следующим образом. Признак является переменной перечислимого типа с конечным множеством значений. Каждому конкретному значению признака соответствует определенный вид записи. Поэтому вместо одной вариантной записи рассматривается семейство фиксированных записей для каждого значения признака. Семейство конечное, так как конечно число элементов перечислимого типа. Анализ вариантной записи

будет сведен к перебору полученного семейства и обработке конкретной фиксированной записи. Поэтому в дальнейшем без снижения общности результатов будут рассматриваться только фиксированные записи.

3.5.3. ДОПОЛНИТЕЛЬНЫЕ СТРУКТУРНЫЕ ТИПЫ ДАННЫХ

Выше были рассмотрены основные структурные типы данных — массивы и записи, которые встречаются в большинстве ЯП. Кроме них находят применение и другие: множества, объединения, динамические объекты данных, списки, последовательности, стеки, деревья и др. Некоторые из этих типов являются стандартными в конкретных ЯП, другие реализуются путем программного моделирования соответствующих структур и операций над ними. В данной работе для дополнительных структурных типов приводится информация описательного характера и детальный анализ не производится. Это вызвано следующими причинами:

1. Некоторые типы являются собственностью одного или малого числа ЯП и не имеют аналогов в других ЯП.

2. Анализ некоторых типов, моделируемых средствами определенного ЯП, может быть сведен к анализу базовых типов, подробно рассмотренных в данной работе.

3. Реализация некоторых типов и выполнение операций над ними недостаточно формализованы.

Необходимо отметить, что чем сложнее тип данных, тем разнообразнее операции над объектами этого типа и тем менее формализовано представление этих операций в ЯП.

Множества

Примером ЯП, в котором реализован аппарат множеств, является Паскаль [48]. Общая форма записи типа данных *множество* следующая:

$$\text{type } T = \text{powerset } T^0,$$

где T определяет тип множества; T^0 является базовым типом для элементов множества.

Обычно T — перечислимый или целый тип. Для типа T реализованы все основные операции над множествами как математическими объектами — объединение, пересечение, разность, операции включения, определения тождественности и нетождественности. Операции селектора представляют собой выбор элемента типа T^0 из объекта типа T. Операцией конструирования является формирование из одного или нескольких элементов типа T^0 объекта типа T.

Объединения

Общая форма записи для объединения имеет вид [82]

$$\text{type } T = \text{union}\,(T^{v_1}, \ldots, T^{v_n}),$$

где T — тип объединения; $T^{v_1}, \ldots, T^{v_n}$ — базовые типы.

В общем, на базовые типы ограничения не накладываются. Любой объект типа T имеет две компоненты — значение и признак, по которому определяется один из типов T^{v1}, T^{v2}, ..., T^{vn} для данного значения. Механизм реализации объединения подобен механизму реализации вариантных записей. Отличие состоит в том, что сам признак скрыт в отличие от признака вариантной записи, в которую он входит в качестве отдельной компоненты. Все операции над объектами типа аналогичны операциям над вариантными записями.

Динамические объекты данных

Этот тип данных в различных вариантах реализован во многих ЯП: Паскаль [42], Ада [26], ПЛ/1 [115], Си [67] и др. Общая форма записи для этого типа имеет вид

$$\text{type } T = \text{pointer } to\, T^0,$$

где T — определяет ссылочный тип; T^0 — базовый тип.

Базовым может быть любой тип. Объект типа T представляет адрес объекта типа T^0. Фактически ссылочный тип не является структурным, так как переменные этого типа содержат только одно значение, как и объекты простых типов. Однако использование ссылочного типа отличается от использования простых типов. Операции над ссылочными типами не формализованы. Например, язык Паскаль допускает только одну операцию — настройку на элемент базового типа (аналогично операции присваивания). В то же время язык Си допускает над ссылочными типами операции адресной арифметики.

Списки.

Списки являются конструкциями ЯП Лисп [113] или могут реализовываться программным моделированием. Во втором случае элемент списка описывается как запись, содержащая одну или несколько компонент ссылочного типа, которые обеспечивают связь между элементами списка. К последним в этом случае могут быть применены операции, которые аналогичны операциям над фиксированными записями. Кроме них существуют операции, применяемые к целому списку: выбор начального элемента, получение остатка списка, соединение списков, их сравнение, инвертирование, поиск элементов (атомов) в списке и др. Необходимо отметить, что в ЯП Лисп эти операции принадлежат к средствам самого языка. Для языков, не имеющих стандартных средств обработки списков, аналогичные операции должны быть реализованы в виде отдельных процедур. Все это не позволяет определить фиксированное множество стандартных операций над списками, характерное для всех или большинства ЯП.

Последовательности

Общая форма имеет вид [151]

$$\text{type } T = \text{sequence } T^0,$$

где T — тип последовательности; T^0 — базовый тип.

Последовательность является одним из вариантов списка, у которого каждый элемент содержит только одну ссылочную переменную, что обеспечивает одностороннюю связь. Операции над последовательностями аналогичны операциям над списками. Одной из разновидностей последовательности является строка. Для строки каждый элемент кроме ссылочной переменной, содержит компоненты символьного типа. Обработка символьных строк допускается в языке Снобол-4 [126].

Стеки

Стек представляет собой специальным образом организованную память с дисциплиной обработки LIFO (последним прошел — первым обработан). Обычно отдельный элемент стека принадлежит простому типу. Однако, используя средства программного моделирования, можно реализовать обработку стеков, содержащих элементы любых типов. На практике стеки реализуются в виде массивов [16] или списков [154]. В отличие от других дополнительных структурных типов, множество операций над стеками фиксировано и включает: инициализацию стека, занесение элемента в стек, выбор из стека, анализ элемента, находящегося на вершине стека.

Операции над стеками могут быть реализованы на аппаратном уровне, стандартными средствами ЯП или программным моделированием.

Деревья

Деревья являются списковыми структурами и служат для представления графов или других аналогичных объектов [154]. Множество операций над деревьями аналогично множеству операций над списками. Реализация этих операций зависит от конкретных приложений.

Кроме рассмотренных дополнительных структурных типов используются и другие — таблицы, файлы, всевозможные комбинации перечисленных выше типов и др. Полный анализ всех типов данных и их преобразований выходит за рамки настоящей книги. Однако предлагаемый подход может применяться и для использования этих типов.

3.6. МЕТОД ПОСТРОЕНИЯ ОПЕРАЦИЙ ПРЕОБРАЗОВАНИЯ ТИПОВ ДАННЫХ

Выше построено множество Σ, состоящее из шести алгебраических систем: $\sum = \{\mathfrak{U}^b, \mathfrak{U}^c, \mathfrak{U}^i, \mathfrak{U}^r, \mathfrak{U}^a, \mathfrak{U}^z\}$. Следующим этапом в разработке межъязыкового интерфейса, согласно постановке задачи, будет построение изоморфных отображений между элементами множества $\sum$, описывающими типы данных в различных ЯП. Поэтому в общем случае множество $\sum$ будет состоять из нескольких подмножеств:

$$\Sigma = \bigcup_{\alpha=1}^{\eta} \sum_{\alpha}, \tag{3.18}$$

где Σ_α описывает множество типов данных в языке l_α, принадлежащем классу L.

При построении изоморфных отображений между алгебраическими системами необходимо определить их свойства, основанные на сохранении интерпретации однотипных операций в различных системах. Для полноты анализа необходимо рассмотреть все возможные комбинации между элементами каждого из множеств Σ_α и между элементами различных $\sum_\alpha$ и $\sum_\beta$, где $\alpha \neq \beta$. При строгом анализе значительное число отображений между системами $\mathfrak{U}^t_\alpha$ и $\mathfrak{U}^q_\beta$ не рассматривается, что связано с отличиями некоторых множеств Ω^t_α и Ω^q_β и отсутствием изоморфных отображений между определенными основными множествами X^t_α и X^q_β. В связи с этим, чтобы не ограничивать множество операций преобразований типов данных, будут рассмотрены следующие случаи:

1. Алгебраические системы $\mathfrak{U}^t_\alpha$ и $\mathfrak{U}^q_\beta$ изоморфны. Существуют изоморфизмы между X^t_α и X^q_β. Множества Ω^t_α и Ω^q_β совпадают.

2. Существуют изоморфизмы между X^t_α и X^q_β. Множества Ω^t_α и Ω^q_β различные. Если $\Omega^t_\alpha \cap \Omega^q_\beta = \Omega$ и Ω не пусто, то рассматриваются отображения между модифицированными алгебраическими системами $\mathfrak{U}^t_\alpha = \langle X^t_\alpha, \Omega \rangle$ и $\mathfrak{U}'^q_\beta = X^q_\beta, \langle \Omega \rangle$.

3. Между множествами X^t_α и X^q_β отсутствует изоморфное отображение. Рассмотрено несколько частных случаев для преобразования типов данных, принадлежащих этой группе.

Переходя к строгому анализу изоморфных отображений, отметим следующий очевидный факт: мощности алгебраических систем должны быть равны:

$$|\mathfrak{U}^t_\alpha| = |\mathfrak{U}^q_\beta|. \tag{3.19}$$

Под мощностью алгебраической системы $\mathfrak{U}^t_\alpha$ понимается мощность множества X^t_α [6]. Несмотря на тривиальность результата, этот факт может служить очень сильным критерием оценки правильности сопряжения модулей для случая, если в разных ЯП одинаковые типы данных имеют различные множества значений. Другим применением этого критерия может служить перенос программного обеспечения с одного типа ЭВМ на другой с различными архитектурами и диапазонами целых и вещественных чисел. Для получения строгих результатов условие (3.19) всегда должно выполняться.

Анализируя множества Ω^t_α, можно отметить, что они все содержат операции отношения. Поэтому на самих множествах X^t_α задано отношение порядка. Учитывая, что любые два элемента из X^t_α сравнимы, это отношение определяет линейный порядок. В этом случае изоморфное отображение должно сохранять отношение линейного порядка. В дальнейшем всегда будет предполагаться выполнение этого требования.

Пусть Σ — множество алгебраических систем, построенных выше. Тогда имеет место следующая лемма.

Лемма 3.1. *Для любого изоморфного отображения* φ *между алгебраическими системами* $\mathfrak{U}^t_\alpha$ *и* $\mathfrak{U}^q_\beta$ *выполняются равенства* $\varphi(X^t_{\alpha.\min}) = X^q_{\beta.\min}$, $\varphi(X^t_{\alpha.\max}) = X^q_{\beta.\max}$.

Доказательство данной леммы простое. Для всех построенных алгебраических систем, описывающих простые типы данных, множества значений ограниченны (числовые типы рассматриваются как отрезки). Согласно аксиомам (3.9) минимальные и максимальные элементы этих множеств им принадлежат. Структурные типы этих данных строятся из простых с помощью конечного числа операций, а их множества значений также конечны. Поэтому $X^t_{\alpha.\min}$, $X^t_{\alpha.\max}$, $X^q_{\beta.\min}$, $X^q_{\beta.\max}$ существуют и принадлежат множествам X^t_α и X^t_β соответственно. Учитывая линейную упорядоченность этих множеств, необходимо, чтобы выполнялось условие леммы. Если бы оно не выполнялось, то изоморфное отображение не сохраняло бы линейный порядок, что противоречит сделанному ранее выводу. Лемма доказана.

3.6.1. ОПЕРАЦИИ ПРЕОБРАЗОВАНИЯ ПРОСТЫХ ТИПОВ ДАННЫХ

Исследуем следующие типы преобразований:

перечислимый — в перечислимый;

булевый — в булевый;

целый — в целый;

вещественный — в вещественный.

Рассмотрим операцию преобразования между перечислимыми типами на примере символьных типов. Пусть $\mathfrak{U}^c_\alpha$ и $\mathfrak{U}^c_\beta$ описывают два символьных типа в некоторых ЯП. Ранее были получены результаты о том, что изоморфное отображение должно сохранять порядок и $|\mathfrak{U}^c_\alpha| = |\mathfrak{U}^c_\beta|$. Имеет место следующая теорема.

Теорема 3.1. *Пусть* φ *— отображение алгебраической системы* $\mathfrak{U}^c_\alpha$ *в алгебраическую систему* $\mathfrak{U}^c_\beta$. *Для того чтобы отображение* φ *было изоморфным, необходимо и достаточно, чтобы* φ *изоморфно отображало* X^c_α *на* X^c_β *с сохранением линейного порядка.*

Необходимость. Пусть φ — изоморфизм. Тогда при отображении сохраняются все операции множества $\Omega = \Omega^c_\alpha = \Omega^c_\beta$, в том числе и операция отношения, которая определяет линейный порядок на множествах X^c_α и X^c_β.

Достаточность. Пусть φ изоморфно отображает X^c_α на X^c_β с сохранением линейного порядка. Необходимо проверить сохранность операций, указанных в (3.8). Операция отношения выполняется согласно упорядоченности. Рассмотрим операцию succ. Согласно лемме 3.1 $\varphi(X^c_{\alpha.\min}) = X^c_{\beta.\min}$. Последовательно применяя операцию succ к данному равенству и учитывая линейную упорядоченность множества X^c_α и X^c_β ($x < \mathrm{succ}(x)$), мы получим, что для любого $x^c_\alpha \in X^c_\alpha$ и $x^c_\alpha \neq X^c_{\alpha.\max}$ из равенства $\varphi(x^c_\alpha) = x^c_\beta$, где $x^c_\beta \in X^c_\beta$ следует равенство

$$\varphi(\mathrm{succ}(x^c_\alpha)) = \mathrm{succ}(x^c_\beta). \qquad (3.20)$$

Для операции pred рассмотрение аналогичное. Согласно лемме 3.1 $\varphi(X^c_{\alpha.\,\max}) = X^c_{\beta.\,\max}$. Последовательно применяя операцию pred, получим, что для любого $x^c_\alpha \in X^c_\alpha$ и $x^c_\alpha \neq X^c_{\alpha.\,\min}$ из равенства $\varphi(x^c_\alpha) = x^c_\beta$, где $x^c_\beta \in X^c_\beta$, следует равенство

$$\varphi(\mathrm{pred}(x^c_\alpha)) = \mathrm{pred}(x^c_\beta). \tag{3.21}$$

Теорема доказана. Она позволяет использовать в качестве операции преобразования между перечисленными типами любое изоморфное отображение, удовлетворяющее условиям теоремы, независимо от природы элементов множеств X^c_α и X^c_β. Если в вызывающем модуле фактический параметр имеет тип

$$\mathrm{type}\ TV = (\text{'A'},\ \text{'B'},\ \text{'C'}),$$

а в вызываемом модуле формальный параметр описан в виде

$$\mathrm{type}\ TF = (1,\ 2,\ 3),$$

то в качестве операции преобразования можно выбрать любое изоморфное отображение вида

$$\varphi(\text{'A'}) = 1,$$
$$\varphi(\text{'B'}) = 2,$$
$$\varphi(\text{'C'}) = 3.$$

При этом необходимо помнить, что над объектами типа TF можно использовать только операции перечислимого типа, определенные в (3.8).

Рассмотрим операцию преобразования между булевыми типами. Пусть $\mathfrak{U}^b_\alpha$ и $\mathfrak{U}^b_\beta$ — алгебраические системы, описывающие эти типы. Тогда справедлива следующая теорема.

Теорема 3.2. *Любой изоморфизм* φ *между алгебраическими системами* $\mathfrak{U}^b_\alpha$ *и* $\mathfrak{U}^b_\beta$ *является тождественным изоморфизмом:*

$$\begin{aligned} \varphi(X^\beta_{\alpha.\,\mathrm{false}}) &= X^b_{\beta.\,\mathrm{false}}, \\ \varphi(X^b_{\alpha.\,\mathrm{true}}) &= X^b_{\beta.\,\mathrm{true}}, \end{aligned} \tag{3.22}$$

где через $x^\beta_{\alpha.\,\mathrm{false}}(X^b_{\beta.\,\mathrm{false}})$ *и* $x^\beta_{\alpha.\,\mathrm{true}}(X^b_{\beta.\,\mathrm{true}})$ *обозначены соответственно* элементы false *и* true *в множестве* $X^b_\alpha(X^b_\beta)$.

Доказательство. Согласно теореме 3.1 для отображения между $\mathfrak{U}^b_\alpha$ и $\mathfrak{U}^b_\beta$ выполняются все операции из (3.7), кроме операций булевой алгебры. При выполнении последних операций всегда справедливо неравенство $X^\beta_{\alpha.\,\mathrm{false}} < X^b_{\alpha.\,\mathrm{true}}$. Поэтому, учитывая, что φ сохраняет порядок, единственно возможным изоморфизмом является отображение вида (3.22). Теорема доказана.

В общем случае X^β_α и X^b_β — различные множества (например, в ЯП ПЛ/1 в качестве булевых переменных используются битовые строки). Если $X^\beta_\alpha = X^b$, то φ является тождественным автоморфизмом.

Рассмотрим операции преобразования для числовых типов. Введем стандартные обозначения для множеств целых чисел Z, рациональных Q и действительных (вещественных) R. Ранее предполагалось, что основные множества алгебраических систем, соответствующие числовым типам, ограниченны. Для дальнейшего анализа снимем это ограничение и будем считать, что при необходимости границы числового диапазона могут быть расширены.

Рассмотрим алгебраические системы $\mathfrak{U}_\alpha^i$ и $\mathfrak{U}_b^i$, соответствующие целым типам, и изоморфизм φ между ними. Изучим свойства этого изоморфизма. Для операции сложения выполняется равенство $\varphi(x_\alpha^i + y_\alpha^i) = \varphi(x_\alpha^i) + \varphi(y_\alpha^i)$. Полагая $y_\alpha^i = 0$, получаем, что $\varphi(x_\alpha^i) = \varphi(x_\alpha^i) + 0$. Так как x_α^i — произвольное, то данное равенство будет выполняться только при $\varphi(0) = 0$.

Для операции умножения необходимо выполнение равенства $\varphi(x_\alpha^i \cdot y_\alpha^i) = \varphi(x_\alpha^i) \cdot \varphi(y_\alpha^i)$. Полагая $y_\alpha^i = 1$, получаем, что $\varphi(x_\alpha^i) = \varphi(x_\alpha^i) \cdot \varphi(1)$. Здесь $\varphi(x_\alpha^i)$ — целое число. Если оно не равно нулю, то выполним операцию целочисленного деления и получим, что $\varphi(1) = 1$. Окончательно можно записать

$$\varphi(0) = 0, \quad \varphi(1) = 1, \tag{3.23}$$

где φ — изоморфизм между $\mathfrak{U}_\alpha^i$ и $\mathfrak{U}_\beta^i$.

Пусть $X_\alpha^i \subset Z$ и $X_\beta^i \subset Z$. Тогда $\forall x_\alpha^i \in X_\alpha^i \exists x_\beta^i \in X_\beta^i : \varphi(x_\alpha^i) = x_\beta^i$. Рассмотрим произвольное $x_\alpha^i > 1$. Последовательно применяя результаты (3.23), получаем

$$\varphi(x_\alpha^i) = \varphi(x_\alpha^i - 1) + \varphi(1) = \varphi(x_\alpha^i - 1) + 1 = \cdots = \varphi(1) + x_\alpha^i - 1 = x_\alpha^i.$$

Пусть $x_\alpha^i < 0$ и $|x_\alpha^i| = y_\alpha^i > 0$. Отметим следующий результат:

$$0 = \varphi(y_\alpha^i + (-y_\alpha^i)) = \varphi(y_\alpha^i) + \varphi(-y_\alpha^i) = y_\alpha^i + \varphi(-y_\alpha^i).$$

Отсюда следует, что $\varphi(-y_\alpha^i) = -y_\alpha^i$ или $\varphi(x_\alpha^i) = x_\alpha^i$. Окончательно можно отметить, что для любого целого x_α^i необходимо равенство

$$\varphi(x_\alpha^i) = x_\alpha^i. \tag{3.24}$$

Рассмотрим изоморфное отображение φ между алгебраическими системами $\mathfrak{U}_\alpha^r$ и $\mathfrak{U}_\beta^r$, соответствующие вещественным типам, где X_α^r и X_β^r — подмножества R. Так как $Z \subset R$, то результат (3.24) справедлив и для подмножества вещественных чисел, совпадающего с Z. Пусть $x_\alpha^r \in Q$, т. е. $x_a^r = m/n$, где m и n — целые числа. Тогда отметим следующее:

$$m = \varphi(m) = \varphi\left(n \frac{m}{n}\right) = \varphi(n) \cdot \varphi\left(\frac{m}{n}\right) = n \cdot \varphi\left(\frac{m}{n}\right).$$

Отсюда следует равенство

$$\varphi\left(\frac{m}{n}\right) = \frac{m}{n}. \tag{3.25}$$

Это соотношение выполняется для любых рациональных чисел. На основании известной теоремы из теории множеств (см.,например, [6, 28]) любое вещественное число x можно с любой заранее определенной точностью ε представить в виде рационального числа m/n так, что $\left|x - \frac{m}{n}\right| < \varepsilon$. Это тем более справедливо для представления вещественных чисел в ЭВМ, так как точность представления зависит от количества разрядов машинного слова памяти. Таким образом, для любых $x \notin R$ выполняется

$$\varphi(x) = x. \tag{3.26}$$

Учитывая, что множества Z и R являются базовыми для основных множеств X_α^i и X_α^r алгебраических систем $\mathfrak{U}_\alpha^i$ и $\mathfrak{U}_\alpha^r$, определенных в п. 2.4, мы доказали следующую теорему.

Теорема 3. 3. *Любой изоморфизм между алгебраическими системами соответствующими числовым типам, является тождественным автоморфизмом.*

В доказательстве использовался тот факт, что 0 и 1 принадлежат основным множествам. Это существенное предположение, так как некоторые числовые отрезки не содержат этих значений. Такие типы данных должны анализироваться в частном порядке.

Выше получены результаты, следующие из анализа изоморфных отображений алгебраических систем. Перейдем к рассмотрению случая изоморфности основных множеств X_α^i и X_β^q при условии отличия Ω_α^i и Ω_β^q. Пусть $\Omega_\alpha^i \cap \Omega_\beta^q \neq 0$. Возможны следующие виды преобразований между типами данных:

символьный — в целый;
символьный — в булевый;
целый — в символьный;
целый — в булевый;
булевый — в целый;
булевый — в символьный.

В приведенных преобразованиях отсутствует вещественный тип. Это связано с тем, что множества значений символьных, булевых и целых чисел имеют дискретный характер, множество вещественных чисел по своей сути непрерывно (при реализации на ЭВМ это множество также дискретно, что связано с особенностями структуры основной памяти).

Рассмотрим преобразования **символьного** типа в **целый** и **целого в символьной.** Пусть $\mathfrak{U}_\alpha^c$ и $\mathfrak{U}_\beta^i$ — алгебраические системы, описывающие эти типы. Пересечение множеств операций $\Omega = \Omega_\alpha^c \cap \Omega_\beta^i$ имеет вид

$$\Omega = \{\text{pred}, \text{ succ}, \leqslant\}. \tag{3.27}$$

Но оно в точности совпадает со множеством операций для любого перечислимого типа. Поэтому мы можем воспользоваться результатами теоремы 3.1 и выбрать любое изоморфное отображение множеств X^c_α и X^i_β, сохраняющее линейный порядок. В качестве множества X^i_β может быть выбран любой отрезок целого типа, для которого

$$|X^c_\alpha| = X^i_{\beta.\max} - X^i_{\beta.\min} + 1, \qquad (3.28)$$

т. е. должно выполняться условие (3.19) о равенстве мощностей основных множеств для алгебраических систем. Особый интерес представляют отображение, ставящее каждому символу его порядковый номер в алфавите (функция ord), и отображение, определяющее по порядковому номеру символа в алфавите его значение (функция chr). В этом случае функции ord и chr являются операциями преобразования между символьным и целым типами.

Рассмотрим преобразование **целого в булевый** и **булевого в целый**. Пусть алгебраические системы $\mathfrak{U}^i_\alpha$ и $\mathfrak{U}^b_\beta$ описывают соответственно целый и булевый типы. Пересечение множеств операций $\Omega = \Omega^i_\alpha \cap \cap \Omega^b_\beta$ совпадает с (3.27). Поэтому в данном случае применимы предыдущие результаты и над целым и булевым типами возможны только операции как над перечислимыми типами.

Рассмотрим преобразование **символьного** в **булевый** и **булевого** в **символьный**. Практического применения эти преобразования не имеют. Однако для них справедливы результаты, как и в случае преобразования между символьными и целыми типами. Возможно только применение операций для перечислимых типов. Множества значений каждого из типов для данного случая будут содержать по 2 элемента.

3.6.2. ОПЕРАЦИИ ПРЕОБРАЗОВАНИЯ СТРУКТУРНЫХ ТИПОВ ДАННЫХ

Рассмотрим операции преобразования для массивов и записей. Анализируя множества операций для соответствующих алгебраических систем (3.15) и (3.17), можно отметить, что при преобразовании должен сохраняться только линейный порядок.

Возможны следующие виды преобразований: массив — в массив, запись — в запись, массив — в запись, запись — в массив.

Рассмотрим первое из них. Пусть $\mathfrak{U}^a_\alpha$ и $\mathfrak{U}^a_\beta$ — две алгебраические системы, описывающие массивы. Пусть φ — их изоморфизм, сохраняющий линейный порядок. Это означает, что

$$\begin{gathered}(\forall x^a_{\alpha_1} \in X^a_\alpha) \,\&\, (\forall x^a_{\alpha_2} \in X^a_\alpha) \,\&\\ (x^a_{\alpha_1} \leqslant x^a_{\alpha_2}) \Rightarrow \varphi(x^a_{\alpha_1}) \leqslant \varphi(x^a_{\alpha_2}).\end{gathered} \qquad (3.29)$$

Согласно (3.16), $x^a_{\alpha_1}$ и $x^a_{\alpha_2}$ — отображения (функции). Выполнение отношения $\leqslant$ для функций означает выполнение этого отношения для всех элементов области определения этих функций.

Рассмотрим определение выражения для $\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$. Из (3.16) следует

$$\begin{aligned} x^a_{\alpha_1} &: I^a_\alpha \to Y(x^a_{\alpha_1}), \quad Y(x^a_{\alpha_1}) \subset \overline{Y}(X^a_\alpha), \\ x^a_{\alpha_2} &: I^a_\alpha \to Y(x^a_{\alpha_2}), \quad Y(x^a_{\alpha_2}) \subset \overline{Y}(X^a_\alpha). \end{aligned} \tag{3.30}$$

Пусть выполняется

$$\varphi_I : I^a_\alpha \to I^a_\beta, \quad \varphi_Y : \overline{Y}(X^a_\alpha) \to \overline{Y}(X^a_\beta), \tag{3.31}$$

где φ_I и φ_Y — изоморфные преобразования соответствующих множеств. Через E_1 обозначим вложение $Y(x^a_{\alpha_1}) \to Y(X^a_\alpha)$, а через E_2 — вложение $Y(x^a_{\alpha_2}) \to \overline{Y}(X^a_\alpha)$. Тогда $\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$ будут определяться как ограничения отображения φ_Y на соответствующие подмножества и обозначаться через $\varphi_Y | Y(x^a_{\alpha_1})$ и $\varphi_Y | Y(x^a_{\alpha_2})$ соответственно. Последнее неравенство в (3.29) будет эквивалентно

$$\varphi_Y | Y(x^a_{\alpha_1}) \leqslant \varphi_Y | Y(x^a_{\alpha_2}). \tag{3.32}$$

Расширим смысл обозначений в (3.31). Пусть теперь φ_I и φ_Y будут обозначать изоморфные отображения между алгебраическими системами, для которых I^a_α, I^a_β, $\overline{Y}(X^a_\alpha)$, $\overline{Y}(X^a_\beta)$ являются основными множествами. Тогда имеет место следующая теорема.

Теорема 3.4. *Пусть* $\mathfrak{U}^a_\alpha$ *и* $\mathfrak{U}^a_\beta$ *— двое алгебраические системы, соответствующие типам данных массив, и* φ_I, φ_Y *— изоморфные отображения, сохраняющие линейный порядок и определяемые* (3.31). *Тогда изоморфизм* φ *между* $\mathfrak{U}^a_\alpha$ *и* $\mathfrak{U}^a_\alpha$ *и* $\mathfrak{U}^a_\beta$ *полностью определяется отображениями* φ_I *и* φ_Y. *Это означает, что анализ отображения* φ *может быть сведен к анализу* φ_I *и* φ_Y *и значительно упрощается задача построения изоморфного отображения.*

Доказательство. Пусть имеются отображения φ_I и φ_Y. Отображение φ_I сохраняет линейный порядок, и, следовательно, сохраняется взаимная упорядоченность отдельных элементов массивов относительно друг друга. Рассмотрим выражение (3.29). С учетом предыдущего результата выполнение (3.29) достаточно проверить только для одного элемента массива. Пусть $k \in I^a_\alpha$. Ему будет соответствовать $\varphi_I(k) \in I^a_\beta$. Пусть выполняется неравенство $x^a_{\alpha_1}(k) \leqslant x^a_{\alpha_2}(k)$. Применим к обеим частям неравенства отображение φ_Y в смысле (3.32). Упорядоченность не нарушается при применении операций конструирования структурных типов. Поэтому независимо от типа данных для множеств $\overline{Y}(X^a_\alpha)$ и $\overline{Y}(X^a_\beta)$ отображение φ_Y также сохраняет линейный порядок. Следовательно, справедливо неравенство

$$\varphi_Y | Y(x^a_{\alpha_1}(k)) \leqslant \varphi_Y | Y(x^a_{\alpha_2}(k)),$$

при этом каждому значению $\varphi_I(k)$ соответствует $\varphi_Y | Y(x^a_\alpha(k))$. Теорема доказана.

Таким образом, можно записать, что φ является двухкомпонентной последовательностью $\varphi = (\varphi_J, \varphi_Y)$. Преобразование массивов определяется преобразованиями типов данных для индексов и множеств значений элементов массивов, принадлежащих к рассматриваемому типу.

Рассмотрим преобразование типа запись — в запись. Пусть $\mathfrak{U}^z_\alpha$ и $\mathfrak{U}^z_\beta$ — две алгебраические системы, описывающие записи. В теории структурной организации данных [151] множество значений для записей определяется как прямое произведение множеств значений их отдельных компонент. Данный подход к описанию записей содержит существенный недостаток. Рассмотрим множество значений алгебраической системы (3.17). Пусть $n = 3$ и имеется три описания:

а)

$$\text{type } T^{z_1} = (S_{v_1} : T^{v_1};\quad S_{v_2} : T^{v_2};\quad S_{v_3} : T^{v_3});$$

б)

$$\text{type } T^{z_2} = (S_{a_1} : T^{v_1};\quad S' : T'),$$
$$\text{type } T' = (S_{v_2} : T^{v_2};\quad S_{v_3} : T^{v_3});$$

в)

$$\text{type } T^{z_3} = (S' : T';\quad S_{v_3} : T^{v_3});$$
$$\text{type } T' = (T_{v_1} : T^{v_1};\quad S_{v_2} : T^{v_2}).$$

Несмотря на различия в описании трех типов записей T^{z_1}, T^{z_2}, T^{z_3}, множества их значений совпадают относительно задачи преобразования данных. Поэтому имеется неопределенность при реализации операции преобразования вида запись — в запись. Для устранения этой неопределенности введем дополнительное ограничение, которое состоит в том, что между множествами типов данных обеих записей можно установить взаимно однозначное соответствие. Из этого ограничения следует:

количество компонент в обеих записях одинаково;

для каждой из компонент первой записи существует только одна соответствующая компонента второй записи.

Операции отношения над записями будут выполняться с учетом выбранного соответствия (последовательности сравниваемых компонент определяются данным соответствием). В этом случае анализ преобразования записи сводится к анализу преобразований типов отдельных компонент. Если изоморфные отображения между отдельными компонентами сохраняют линейный порядок для самих компонент, то с учетом сказанного выше мы доказали следующую теорему.

Теорема 3.5. *Пусть* $\mathfrak{U}^z_\alpha$ *и* $\mathfrak{U}^z_\beta$ *— две алгебраические системы, соответствующие типам данных* **запись**, *и любые* $x^z_\alpha \in X^z_\beta$, $x^z_\beta \in X^z_\beta$. *Тогда если между последовательностями компонент* x^z_α *и* x^z_β *существует взаимно однозначное соответствие, то изоморфизм между* $\mathfrak{U}^z_\alpha$ *и* $\mathfrak{U}^z_\beta$ *определяется изоморфными отображениями алгебраических систем, соответствующих компонентам записей.*

Рассмотрим смешанные виды преобразований между структурными типами. Пусть $\mathfrak{U}^a_\alpha$ и $\mathfrak{U}^z_\beta$ — алгебраические системы, описывающие массив и запись соответственно. Множества операций для этих систем совпадают. Поэтому определим условия изоморфного отображения для множеств значений этих типов. Как и в случае преобразования записей, для данного вида преобразования имеется некоторая неопределенность, которую можно устранить, введя следующее ограничение: количество элементов массива должно совпадать с количеством компонент записи. Тогда массив рассматривается как запись, у которой множество селекторов совпадает со множеством индексов, а типы всех компонент одинаковы. Поэтому для преобразования вида массив — в запись можно использовать результаты теоремы 3.5.

Преобразование вида запись — в массив требует, чтобы типы всех компонент записи были одинаковыми, а сама запись была представлена в виде массива. Множество значений компонентов образует множество значений элементов массива Y. Множество индексов строится простым переупорядочиванием селекторов компонентов записи: i-й компоненте записи ставится в соответствие $\varphi(i)$-й элемент во множестве J (φ — взаимно-однозначное отображение. Само множество I будет множеством значений перечислимого типа. В этом случае можно использовать результаты теоремы 3.4.

3.7. ОПЕРАЦИИ ИЗМЕНЕНИЯ УРОВНЯ СТРУКТУРИРОВАНИЯ ДАННЫХ

К этим операциям относятся операции селектора и конструирования. Операции селектора обеспечивают выбор из объекта структурного типа отдельных компонент: для массива — отдельные элементы, для записи — компоненты записи. В п. 3.5 при анализе структурных типов данных дано определение этих операций. Операция выбора элемента массива определяется как ограничение $x\,|\,\{k\}$, где x — объект типа массив, представленный отображением (3.16), а k — элемент из множества индексов. Введем обозначение для этой операции

$$S^a = S^a(x,\ K). \tag{3.33}$$

Аналогично была определена операция селектора для записей как ограничение $x\,|\,\{S_{v_m}\}$, где x определяет соответствие между множеством селекторов записи $I = \{S_{v_1},\ \ldots,\ S_{v_n}\}$ и компонент записи $X^v = \{x^{v_1},\ \ldots,\ x^{v_n}\}$, а S_{v_m} — селектор выбираемой компоненты. Введем обозначение для этой операции

$$S^z = S^z(x,\ S_{v_m}). \tag{3.34}$$

Такая интерпретация операций селектора позволяет не учитывать типы отдельных элементов и компонент структурных типов, а рассматривать их как множество объектов произвольной природы. Окончательно множество операций селектора будет иметь вид

$$S = \bigcup_{\alpha=1}^{n} S_\alpha, \quad S_\alpha = \{S^a_\alpha,\ S^z_\alpha\}, \tag{3.35}$$

где S_α — множество операций селектора для структурных типов в ЯП l_α; S^a_α — операция селектора для массивов в ЯП l_α; S^z_α — операция селектора для записей в ЯП l_α.

Определим операции конструирования структурных типов данных. Как и для операций селектора, будем полагать, что множество объектов, из которых конструируется структурный тип, имеет произвольную природу. Для конструирования типа *массив* необходимо, чтобы все его элементы имели одинаковый тип. В этом случае для операции конструирования необходимо:

упорядочить множество элементов;

построить множество индексов для элементов массива;

установить однозначное отображение между множеством индексов и множеством элементов.

Пусть имеется конечное множество объектов $X' = \{x_1, \ldots, x_n\}$ некоторого типа T^t. Запишем операцию конструирования массива:

$$C^a = C^a(x_1, \ldots, x_n). \tag{3.36}$$

Множество элементов упорядочено естественным образом в порядке их следования в описании (при необходимости естественную упорядоченность можно изменить). Множество индексов будет иметь вид $I = (1, \ldots, n)$, что соответствует множеству значений перечислимого или отрезка целого типа. Отображение между множествами I и X' определяется тем, что каждому $k \in I$ соответствует $x_k \in X'$. Обозначим это отображение через x^a. Окончательно получим

$$x^a : I \to Y(x^a),$$

где $Y(x^a)$ определяет множество значений элементов $x_1, \ldots, x_n$. При этом $Y(x^a) \subset X^t$ (X^t — множество значений для типа T^t). Этот массив может быть описан следующим образом:

$$\text{type } T^a = \text{array } T(I) \text{ of } T^t,$$

где T^a — тип данных массива; $T(I)$ — тип данных для множества индексов (перечислимый или отрезок целого типа).

Определим операцию конструирования записи. Метод построения аналогичен методу построения массивов. Имеется конечное множество объектов $X^v = \{x^{v_1}, \ldots, x^{v_n}\}$ типов $T^{v_1}, \ldots, T^{v_n}$ соответственно. Операцию конструирования записи представим в виде

$$C^z = C^z(x^{v_1}, \ldots, x^{v_n}). \tag{3.37}$$

Множество элементов упорядочено естественным образом в порядке их следования в описании операции (при необходимости естественную упорядоченность можно изменить). Каждой компоненте $x^v{}_m$ поставим в соответствие селектор S_{v_m}, который может быть номером компоненты в списке (3.37). Этим полностью определяется операция конструирования записи

$$\text{type } T^z = (S_{v_1} : T^{v_1}; \ \ldots; S_{v_n} : T^{v_n}),$$

где T^z — тип записи.

Окончательно запишем множество операций конструирования:

$$C = \bigcup_{\alpha=1}^{n} C_\alpha, \quad C_\alpha = \{C_\alpha^a, C_\alpha^z\}, \tag{3.38}$$

где C_α — множество операций конструирования для структурных типов в ЯП; C_α^a — оператор конструирования массива в l_α; C_α^z — оператор конструирования записи в l_α.

3.8. ПРАКТИЧЕСКИЕ ВОПРОСЫ ПОСТРОЕНИЯ МЕЖМОДУЛЬНЫХ ИНТЕРФЕЙСОВ

Проведенный выше анализ определяет необходимые условия, методы, правила и т. д., служащие основой построения межъязыкового интерфейса. Однако на практике часто встречаются задачи сопряжения модулей, которые не могут быть рассмотрены в рамках приведенных моделей, методов и средств. Для их решения необходима разработка методов, отличных от рассмотренных. Они обычно содержат допущения, противоречающие условиям формального анализа, — нарушение свойств модулей, изоморфности алгебраических систем и т. д.

Случаи нарушения условий построения МЯИ

1. Типы данных, описанные в п. 3.5, являются наиболее общими. При разработке модулей часто приходится вводить такие типы, множества значений которых являются подмножествами множеств значений для этих типов. Рассмотренный подход может применяться и для их анализа. Однако результаты в этом случае будут другими. Проиллюстрируем это на примере отрезков числовых типов.

Предыдущий анализ над множествами значений для числовых типов обязательно включает 0 и 1. В некоторых случаях отрезки типов могут не содержать этих значений. Поэтому результаты предыдущего анализа для числовых типов использоваться не могут. Однако выполняемые операции для объектов данных типов могут быть корректными за счет неявных дополнительных условий. Например, для отрезка целого типа вида $m..n$, где $n > m > 1$, могут не применяться операции вычитания div и mod, а операции сложения и умножения будут давать корректные результаты. Данную особенность можно легко объяснить. Алгебраическая система для целого типа (3.10) характеризуется множеством Ω^l, включающим все возможные операции для целого типа. Исключение из этого множества некоторых операций дает нам совершенно иную алгебраическую систему, и преобразование таких типов приводит к другим результатам. Однако в модулях соответствующий тип будет описан как отрезок целого, поэтому для аналогичных случаев анализ преобразования типов должен проводиться в частном порядке. Этот пример иллюстрирует проблему несоответствия в описаниях типов, относящуюся к классу проблем сопряжения, связанных с отличиями в описаниях модулей.

2. Другим аспектом практической реализации может служить несоответствие областей значений одинаковых типов фактического

и формального параметров. Пусть фактический параметр имеет вещественный тип с областью значений вида — $a..a$, а формальный — вещественный тип с областью значений — $b..a$, где $a > b > 0$. Несмотря на различие в множествах значений, операции над объектами будут корректными, если значения объектов и результатов операций будут принадлежать пересечению рассмотренных отрезков. В этом случае при построении алгебраических систем необходимо в качестве множества значений рассматривать пересечение отрезков, чтобы обеспечить построение изоморфного отображения.

3. При анализе преобразований между простыми типами было отмечено, что с вещественными типами возможны только преобразования вида *вещественный — в вещественный*. Это очень сильное ограничение, так как часто в практике программирования возникает преобразование типов вида *целый — в вещественный* и наоборот. Строгий анализ таких преобразований не может быть проведен ввиду отсутствия изоморфного соответствия между множествами значений этих типов — одному целому числу будет соответствовать некоторое множество вещественных чисел. Анализ преобразований для этих типов должен производиться следующим образом. Отображения между множествами целых и вещественных чисел являются частичными мультиотображениями. Пусть φ — отображение множества X^r на X^i, где X^r и X^i — множества вещественных и целых чисел соответственно. Для каждого $x^i \in X^i$ определим прообраз $\varphi^{-1}(x^i) \subset X^r$. Различным x^i будут соответствовать различные прообразы. Введем фактор-множество X^r/φ, элементами которого являются прообразы элементов x^i. В алгебраической системе $\mathfrak{U}^r$, описывающей вещественный тип, заменим множество X^r фактор-множеством X^r/φ. В этом случае отображение между алгебраическими системами сохраняет линейный порядок, но корректность арифметических операций не выполняется. Так, если преобразование вещественного числа в целое состоит в отбрасывании дробной части, то $1{,}6 + 1{,}6 = 3{,}2$ и $\varphi(3{,}2) = 3$. В то же время $\varphi(1{,}6) + \varphi(1{,}6) = 1 + 1 = 2 \neq 3$. Эти особенности необходимо учитывать при практической реализации подобных преобразований.

Данный пример относится к анализу преобразований, при которых отсутствует изоморфное соответствие между основными множествами алгебраических систем, описывающих преобразуемые типы. Анализ подобных преобразований должен проводиться в частном порядке.

4. Среди преобразований, рассмотренных в п. 3.7, не рассматривался случай, когда строка символов, содержащая последовательность цифр, должна преобразовываться в объект целого типа с соответствующим значением.

Как известно, в теории структурной организации данных строка символов описывается в виде массива символьных элементов. Поэтому такое преобразование переводит весь массив (структурный тип) в целое число (простой тип). Аналогичные преобразования, связанные с отображениями структурного типа в простой и наоборот, не входят в задачи межъязыкового интерфейса и поэтому не рассматриваются.

Четыре приведенных выше примера показывают многообразие

проблем и задач сопряжения модулей. Часть из них удается свести к строгому анализу с помощью дополнительных ограничений. Другие задачи не входят в состав функций межъязыкового интерфейса. Однако рассматриваемый подход с соответствующими дополнениями может использоваться для построения любых межъязыковых интерфейсов, предназначенных для комплексирования модулей.

Формальный подход к построению межъязыковых интерфейсов, рассмотренный в данной главе, позволяет упорядочить разработку МЯИ, выделить основные этапы его создания и особые ситуации, возникающие при его построении.

Аспекты практической реализации МЯИ.

1. Выделены два основных этапа в построении МЯИ:

разработка множества интерфейсных функций (операций преобразования, селектора, конструирования);

реализация сопряжения для каждой пары модулей.

Данные этапы в значительной мере независимы. Поэтому практическая реализация соответствующих программных компонент может осуществляться параллельно. Кроме того, фиксированное множество интерфейсных функций может использоваться в различных алгоритмах сопряжения, которые могут соответствовать разным прикладным программным системам. В этом случае множество функций является связующим звеном для данных систем.

2. Формальный подход к интерфейсным функциям позволяет на этапе проектирования выделить множество допустимых операций, исключая операции, которые не могут быть реализованы или не относятся к функциям МЯИ.

3. Результаты формального подхода позволяют осуществлять контроль на совместимость типов передаваемых параметров. Если в вызывающем и вызываемом модулях описаны типы данных фактических и формальных параметров с указанием множеств значений и операций над типами, то на этапе сопряжения можно определять допустимость преобразования типов для соответствующих параметров.

4. Анализ, основанный на формальном подходе, позволяет определить наиболее типичные ошибки сопряжения модулей (при отсутствии МЯИ). К их числу следует отнести ошибки:

несоответствия числа параметров в списках фактических и формальных параметров;

несогласованности типов передаваемых параметров в вызывающем и вызываемом модулях;

несоответствия во множествах значений типов фактических и формальных параметров;

адаптации программного обеспечения на ЭВМ с другой архитектурой (меняются множества значений);

связанные с использованием операций, которые не включены в описания типа данных (чаще всего это характерно для применения разного рода функций);

связанные с различным уровнем структурирования фактических и формальных параметров;

основанные на неверном описании типов данных передаваемых параметров в вызывающем и вызываемом модулях;

вызванные отсутствием обратных преобразований типов данных после работы вызываемого модуля.

Все эти классы ошибок выделены на основании анализа, проведенного в данной главе.

5. Реализация множества интерфейсных функций и алгоритма комплексирования позволяет автоматизировать процесс сопряжения модулей. Такой процесс автоматизированного сопряжения реализован в системе АПРОП, рассмотренной ниже.

В заключение рассмотрим сводку правил сопряжения модулей, обобщающих полученные ранее выводы и результаты.

Правило 1. Списки фактических и формальных параметров должны быть упорядочены — сначала следуют входные, затем выходные параметры. Если какой-либо параметр является и входным и выходным, то он должен присутствовать дважды в соответствующих частях списка.

Правило 2. Провести разбиения списков параметров на подмножества для построения однозначного отображения между списками фактических и формальных параметров. Упорядочить списки после проведения разбиения так, чтобы каждому t-му подмножеству V^t из списка фактических параметров соответствовало t-е подмножество F^t списка формальных параметров.

Для каждого t из списка входных параметров выполнить:

Правило 3. Если $|F^t| > 1$ и $|V^t| = 1$, то следует выбрать из множества операций селектора S необходимую операцию для соответствующей пары ЯП вызывающего и вызываемого модулей. Если операция существует, то следует применить ее к параметру V^t. Если такой операции нет, то необходимо ее построить, включить в состав множества S и применить к параметру V^t. Установить соответствие между каждой компонентой структурного типа (результат применения операции селектора) и соответствующим параметрам из F^t. Изменить отображение между списками параметров, соответствующее входным параметрам, за счет исключения соответствия между V^t и F^t и включения полученных новых соответствий.

Правило 4. Если $|V^t| > 1$ и $|F^t| = 1$, то выбрать из множества операций конструирования C необходимую операцию для соответствующей пары ЯП вызывающего и вызываемого модулей. Если операция существует, то применить ее ко множеству параметров V^t. Если такой операции нет, то необходимо ее построить, включить в состав множества C и применить ко множеству параметров V^t. Изменить отображение между списками параметров, соответствующих входным и выходным параметрам, за счет исключения соответствия между множеством V^t и параметром F^t и включения полученного нового соответствия для структурного типа.

Правило 5. Если $|V^t| > 1$ и $|F^t| > 1$, то данное преобразование не входит в состав задач межъязыкового интерфейса и в настоящей работе не рассматривается. Эта задача решается в частном порядке другими средствами.

После применения правил 3—5 получены модифицированные списки формальных и фактических параметров, они соответствуют входным параметрам, содержащим одинаковое количество элементов, и устанавливают между ними взаимно однозначное соответствие.

Для каждой пары фактических и формальных параметров выполнить:

Правило 6. Из множества P выбрать необходимую операцию преобразования типов данных. Если она существует, то применить ее к данной паре параметров. Если она отсутствует, то построить ее, включить во множество P и применить к рассматриваемой паре параметров. Если построить операцию не удается, то решить данную задачу в частном порядке с помощью других методов.

После обработки входных параметров вызываемым модулем правила 3—6 необходимо применить к выходным параметрам. При этом фактические параметры, а также множества V^t и F^t соответственно меняются местами.

Перечисленные шесть правил определяют содержание последнего этапа построения межъязыкового интерфейса относительно первых двух классов проблем, приведенных в п. 2.1. Необходимо также отметить, что множества операций P, S и C постоянно расширяются за счет включения новых видов преобразований.

3.9. ПРИМЕР РЕАЛИЗАЦИИ МЕЖЪЯЗЫКОВОГО ИНТЕРФЕЙСА

Примером реализации межъязыкового интерфейса, основанным на рассмотренных методах, могут служить соответствующие средства системы АПРОП [30]. Эти средства предназначались для решения следующих задач обеспечения:

перехода от среды функционирования одного ЯП к среде функционирования другого;

передачи управления между разноязыковыми модулями;

доступа к общим данным для разноязыковых модулей;

реализации механизма передачи данных через формальные параметры.

Предварительный анализ, проведенный в данной главе, позволяет выделить следующие этапы в разработке МЯИ: выбор класса языков программирования; определение типов данных в этих ЯП; построение множества операций пребразования типов данных; построение множества операций конструирования и селектора для структурных типов; разработка правил сопряжения пар модулей с использованием построенных множеств операций. Рассмотрим подробнее каждый этап.

Выбор класса ЯП

В класс ЯП межъязыкового интерфейса (МЯИ), реализованного в рамках системы АПРОП, входят ПЛ/1, Фортран, Кобол и Ассемблер ОС ЕС ЭВМ. Выбор этих языков обусловлен их широким распространением, различными сферами применения, и их особенностью является то, что все они не имеют механизмов конструирования новых

типов данных и основные проблемы сопряжения возникают как различия в реализациях систем программирования с ЯП. Результаты исследования показывают, что проблемы сопряжения делятся на проблемы передачи управления и проблемы передачи данных между разноязыковыми модулями.

Предложенные операции МЯИ (преобразования типов данных) конструирования, селектора, передачи управления и передачи данных) реализованы в системе АПРОП в виде макроопределений и загрузочных модулей.

Типы данных ЯП ПЛ/1, Фортран, Кобол, Ассемблер

Существующее множество типов данных в этих ЯП небольшое. Простые типы представлены как предопределенные с небольшими видоизменениями. Структурными типами являются массивы, записи и строки.

При рассмотрении типов данных ЯП, средств их описания и особенностей их представления алгебраическими системами значительное внимание уделяется архитектуре ЕС ЭВМ (структуре памяти, системе команд и др.). В ней под число целого типа отводится слово (4 байта) или полуслово (2 байта) памяти. Числа вещественного типа могут занимать слово или двойное слово (8 байт). Существуют вещественные числа, занимающие 16 байт памяти. Но в практике они встречаются редко и могут быть обработаны только средствами языка Ассемблер. Символьные данные представляются в виде последовательности байт, каждый из которых содержит один символ. Данные булевого типа представляются и обрабатываются как битовые строки. Числа, кроме целого и вещественного типов, могут быть представлены как последовательности десятичных цифр.

Язык ПЛ/1. К простым типам относятся целые (FIXED), вещественные (REAL), десятичные (DECIMAL) числа. Комплексные числа (COMPLEX) представлены в виде прямого произведения и занимают в памяти последовательность из двух чисел.

Символьные и логические данные описываются как строки (байт и бит соответственно). Этот тип данных в интерпретации для ЯП ПЛ/1 не входит в теорию структурной организации данных и занимает промежуточное положение между простым и структурным типами. С точки зрения реализации строки необходимо отнести к структурным, а с точки зрения использования — к простым.

К структурным типам относятся массивы и записи (в терминологии ЯП ПЛ/1 — структуры). Такие типы данных, как базированные переменные, переменные типа метки и др., при построении МЯИ системы АПРОП не рассматривались, хотя могут быть обработаны другими средствами.

Язык Фортран. К простым типам относятся целые (INTEGER) и вещественные (REAL) числа, а также булевые (BOOLEAN) переменные. Комплексные числа (COMPLEX) могут быть представлены последовательностью только из двух вещественных чисел. Символьный тип в языке Фортран отсутствует, однако под расположение символов могут быть отведены переменные любых типов. В Фортране имеется

возможность использовать шестнадцатиричные данные. Как и символьные, они могут быть расположены в переменных любых типов.

Структурные типы языка Фортран представлены только массивами. При этом в памяти они расположены по столбцам в отличие от других ЯП, где массивы располагаются по строкам.

Язык Кобол. Данные простых типов могут быть с фиксированной точкой (в том числе целые), вещественные, десятичные и типа DISPLAY. В последнем случае каждый символ данных занимает 1 байт.

Структурные типы представлены массивами и записями.

Язык Ассемблер. Является машинно-ориентированным языком, и типы данных определяются только в контексте выполняемых команд и не зависят от средств распределения памяти под сами величины. Более подробно типы данных и их сравнительный анализ приведены в [27, 96].

Необходимость преобразования данных для связи разноязыковых модулей обусловлена следующими причинами:

1. Язык ПЛ/1 для представления сложных типов данных (строк, массивов, записей) содержит дискрипторы — информационные векторы (ИВ). Другие ЯП дескрипторов не имеют. При передаче сложных типов данных в ПЛ-модуль необходимо построить ИВ, а из ПЛ-модуля по ИВ определить непосредственный адрес данных. Структуры ИВ приведены в [30].

2. Массивы в Фортран-модулях расположены по столбцам, а для других ЯП — по строкам. Необходимы средства для транспортирования массивов.

3. Отсутствие в ЯП Фортран типа данных *запись* (или структура). При передаче данных такого типа необходимо наличие операций селектора и конструирования.

4. Отсутствие в ЯП Фортран и Кобол типов данных, эквивалентных строкам ЯП ПЛ/1. В некоторых случаях символьные строки могут быть представлены в виде массивов, а битовые — в виде массивов или простых типов данных.

5. Проблема, аналогичная преобразованию строк, возникает при передаче массивов символьных и битовых строк языка ПЛ/1.

6. Компоненты комплексных чисел в ЯП ПЛ/1 могут быть любыми числовыми значениями, в ЯП Фортран — только вещественными; в ЯП Кобол комплексные числа отсутствуют. Возможны передача и обработка комплексных чисел как массива или записи из двух компонент.

7. В некоторых случаях возникает необходимость преобразования числовых величин в различные представления.

Проблемы передачи управления между разноязыковыми модулями вызваны следующими причинами:

а) при входе в ПЛ-модуль или Фортран-модуль необходимо установить среду функционирования для соответствующего ЯП;

б) при сопряжении разноязыковой пары модулей, одним из которых является ПЛ-модуль, необходимо специальным образом осуществлять связь цепочек областей сохранения регистров.

Рассмотрим особенности применения операций над перечислимыми типами данных применительно к рассматриваемым ЯП:

1. Для целого типа множество операций совпадает с Ω^i (см. гл.2), кроме операции mod, которая может быть выражена через остальные.

2. Для вещественного типа множество операций совпадает с Ω^r.

3. Для символьного и булевого типов отсутствуют операции pred и succ.

4. При обработке массивов допустимо использование только отдельных элементов. В ЯП ПЛ/1 возможны операции «+» и «$\times$» для одномерных и двухмерных массивов, интерпретируемые как соответствующие операции над векторами и матрицами.

5. При обработке записей допустимо использование отдельных компонент и в особых случаях всей записи.

6. Множество операций над строками в языке ПЛ/1 включает операции отношения и выбора подстроки.

Необходимо отметить особенности ЯП ПЛ/1, Фортран, Кобол и Ассемблер, упрощающие процесс информационного сопряжения модулей:

а) для соответствующих простых типов данных в разных ЯП множества значений совпадают и определяются размером памяти, отводимой под переменные этих типов;

б) множества значений индексов массивов могут быть только отрезками целых типов;

в) в рассматриваемых языках отсутствует полный контроль типов операндов в операциях.

Это позволяет, например, для символьной строки ЯП ПЛ/1 использовать описание любого массива ЯП Фортран с соблюдением необходимого соотношения длины строки и размера массива.

Проведенный анализ типов данных ЯП ОС ЕС включает перечень типов данных ЯП; особенности их представления; особенности выполнения операций над объектами рассмотренных типов; условия и особенности сопряжения разноязыковых модулей.

Полученные данные были использованы для разработки программных средств реализации функций МЯИ системы АПРОП.

Программные средства реализации функций межъязыкового интерфейса.

Программные средства МЯИ состоят из набора макроопределений и загрузочных модулей, предназначенных для сопряжения разноязыковых модулей по передаче управления и данных. Эти ПС образуют библиотеку МЯИ. Подробное описание этих средств приведено в [96].

Глава 4

МЕТОДЫ УПРАВЛЕНИЯ МОДУЛЬНЫМИ СТРУКТУРАМИ

На основе анализа и определения задач управления модульными структурами выделено одно общее свойство — связь с операциями над модульными структурами программ. Взяв это свойство за основу межмодульного интерфейса, определим окончательный перечень задач в управлении модульными структурами:

1) выполнение операций над модульными структурами;

2) построение программных структур на основе их модульных структур;

3) построение отладочной среды модульной структуры;

4) сборка модулей в единый агрегат с учетом решения задач 1—3.

При решении первых трех задач четвертая является тривиальной, и ее обычно выполняет специальная компонента операционной системы — редактор связей в ОС ЕС, построитель задач в ОС РВ для СМ ЭВМ, комплексатор для МВК «Эльбрус» и т. д. Описание этого процесса можно найти в [10]. Поэтому подробно рассмотрим только три первые задачи.

4.1. ОПРЕДЕЛЕНИЕ МОДУЛЬНОЙ СТРУКТУРЫ ПРОГРАММНЫХ АГРЕГАТОВ

Для представления модульных структур используется математический аппарат теории графов, в котором графом G называется пара объектов $G = (X, \Gamma)$, где X — конечное множество, называемое множеством вершин, а Γ — конечное подмножество прямого произведения $X \times X \times Z$, называемое множеством дуг графа [154]. Из данного определения следует, что граф G фактически является мультиграфом, так как две его вершины могут быть соединены несколькими дугами. Для отличия таких дуг вводится их нумерация целыми положительными числами.

Программная структура, состоящая из модулей (программный агрегат), описывается ориентированным графом. В нем каждая дуга соответствует оператору CALL и соединяет вершину, соответствующую вызывающему модулю, с вершиной, соответствующей вызываемому модулю. Для формального подхода к управлению модульными структурами введем несколько определений.

Определение 4.1. Модель модульной структуры программного агрегата — это объект, описываемый тройкой $T = (G, Y, F)$, где $G = (X, \Gamma)$ — ориентированный граф, являющийся графом модульной структуры; Y — множество модулей, входящих в программный агрегат; F — функция соответствия, ставящая каждой вершине X-графа элемент множества Y.

Функция F отображает X на Y:

$$F : X \to Y. \tag{4.1}$$

В общем случае элементу из Y может соответствовать несколько вершин из X (данная ситуация, как показано ниже, характерна для динамической структуры программного агрегата).

Определение 4.2. Модульной структурой программного агрегата называется пара $S = (T, \chi)$, где T — модель модульной структуры программного агрегата; χ — характеристическая функция, определенная на множестве вершин X графа модульной структуры G.

Значение функции χ определяется следующим образом:

$f(x) = 1$, если модуль, соответствующий вершине $x \in X$, включен в состав программного агрегата;

$f(x) = 0$, если модуль, соответствующий вершине $x \in X$, не включен в состав программного агрегата, но к нему имеются обращения из других модулей, ранее включенных.

Определение 4.3. Две модели модульных структур $T_1 = (G_1, Y_1, F_1)$ и $T_2 = (G_2, Y_2, F_2)$ тождественны, если $G_1 = G_2$, $Y_1 = Y_2$, $F_1 = F_2$. Модель T_1 изоморфна модели T_2, если $G_1 = G_2$, между множествами Y_1 и Y_2 существует изоморфизм φ, а для любого $x \in X \Rightarrow F_2(x) = \varphi(F_1(x))$.

Определение 4.4. Две модульные структуры $S_1 = (T_1, \chi_1)$ и $S_2 = (T_2, \chi)$ тождественны, если $T_1 = T_2$ и $\chi_1 = \chi_2$. Две модульные структуры S_1 и S_2 называются изоморфными, если T_1 изоморфна T_2 и $\chi_1 = \chi_2$.

Понятие изоморфности модульных структур и их моделей необходимо для введения уровня абстракции, на котором определяются операции над модульными структурами. Для изоморфных объектов операции будут интерпретироваться одинаково без ориентации на конкретный модульный состав. В этом случае данные операции определяются над парами (G, χ).

Введенные выше определения описывают модульные структуры общего вида. В настоящей работе рассматриваются структуры специального вида, особенности которых состоят в следующем:

1) граф модульной структуры G имеет одну или несколько компонент связности, каждая из которых представляет ациклический граф, т. е. не содержит ориентированных циклов;

2) в каждой компоненте выделена единственная вершина, которая называется корневой и характеризуется тем, что не существует входящих в нее дуг и соответствующий ей модуль программного агрегата выполняется первым (для данной компоненты связности);

3) циклы допускаются только для случая, когда соответствующий некоторой вершине модуль имеет рекурсивное обращение к самому

себе. Обычно такая возможность реализуется компилятором с соответствующего ЯП и данный тип связи не рассматривается межмодульным интерфейсом. Поэтому такие дуги не включаются в граф. Исключение из рассмотрения других типов циклов связано с тем, что некоторые модули должны будут помнить историю своих вызовов, чтобы правильно вернуть управление. А это противоречит свойствам модулей, рассмотренным в гл. 3;

4) пустой граф G_0 соответствует пустой модульной структуре.

В дальнейшем под терминами *модульная структура*, *граф модульной структуры* и т. д. будут пониматься объекты, удовлетворяющие указанным выше условиям. Типичный пример графа модульной структуры приведен на рис. 4.1.

Вершины x_1, x_2, ..., x_8 составляют множество X. Все дуги пронумерованы. Из модуля, соответствующего вершине x_5, имеются два обращения (два оператора вызова) к модулю, соответствующему вершине x_8. Множество дуг графа имеет вид $\Gamma = \{(x_1, x_2, 1), (x_1, x_3, 1), \ldots, (x_5, x_8, 1), (x_5, x_8, 2)\}$. В дальнейшем этот граф будет использоваться для иллюстрации операций над модульными структурами.

4.2. ТИПЫ ПРОГРАММНЫХ АГРЕГАТОВ

Ранее отмечено, что одной из функций межмодульного интерфейса является комплексирование различных типов программных агрегатов. Любой из них на различных этапах обработки характеризуется двумя признаками: завершенностью построения модульной структуры агрегата и признаком подчиненности. Первый признак определяет включение всех модулей в структуру. Построение модульной структуры не окончено, если имеются вершины графа с нулевым значением характеристической функции. Второй признак определяет, является ли данный программный агрегат самостоятельным объектом для выполнения или его модульная структура входит как составная часть в модульную структуру более высокого уровня.

Введем две функции согласно указанным признакам. Для признака завершенности построения функция $C(S)$ определяется следующим образом:

$C(S) = 1$, если $\chi(x) = 1$ для любых x из X;

$C(S) = 0$, если существует x такое, что $\chi(x) = 0$.

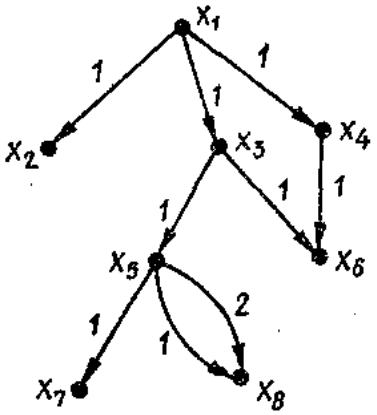


Рис. 4.1. Граф модульной структуры

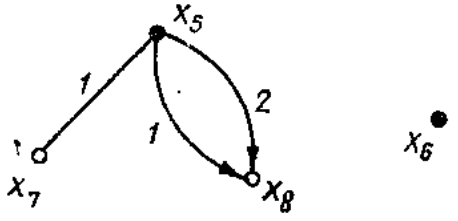


Рис. 4.2. Графы модульных структур, соответствующие модулям x_5 и x_6

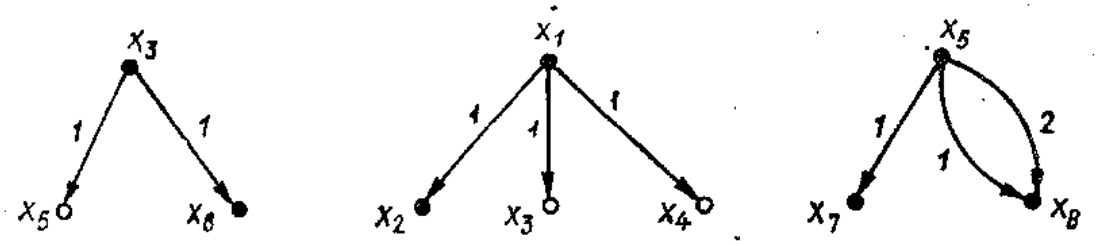


Рис. 4.3. Графы модульных структур для трех видов сегментов

Для признака подчиненности функция $R(S)$ определяется так: $R(S) = 1$, если соответствующий программный агрегат готов к выполнению; $R(S) = 0$, если соответствующая модульная структура входит в состав структуры более высокого уровня.

Определим следующие типы программных агрегатов на основе введенных ранее определений.

Модуль. Модуль является программным агрегатом с графом модульной структуры $G^m = (X^m, \Gamma^m)$ с единственной вершиной $x_i \in X^m$, для которой $\chi(x_i) = 1$. Данная вершина является корневой. Дуга вида (x_j, x_e, k), если она существует, означает обращение из модуля, соответствующего вершине x_j, к модулю, соответствующему вершине x_l. На рис. 4.2 приведены графы двух модульных структур, соответствующие модулям x_5 и x_6.

Темный кружок соответствует вершине, для которой $\chi(x) = 1$; светлый — $\chi(x) = 0$.

Сегмент. Сегмент является программным агрегатом с графом модульной структуры $G^s = (X^s, \Gamma^s)$, для которого выполняется одно из двух условий:

$$C(S^s) = 0, \quad C(S^s) = 1 \text{ и } R(S^s) = 0.$$

В зависимости от комбинации C и R различаются следующие виды сегментов: открытый сегмент ($C = 0$, $R = 0$); сегмент, замкнутый сверху ($C = 0$, $R = 1$); сегмент, замкнутый снизу ($C = 1$, $R = 0$). На рис. 4.3 приведены графы модульных структур для трех видов сегментов соответственно.

Программа. Программа является агрегатом с графом модульной структуры $G^p = (X^p, \Gamma^p)$, для которого выполняется $C(S^p) = 1$; $R(S^p) = 1$. Пример графа модульной структуры для программы приведен на рис. 4.1.

Комплекс. Комплекс является программным агрегатом с графом модульной структуры $G^c = (X^c, \Gamma^c)$, состоящим из n компонент связности ($n > 1$), каждая из которых является графом модульной структуры для соответствующей программы. Для комплекса выполняются

$$G^c = G_1^p \cup G_2^p \cup \cdots \cup G_n^p, \text{ где } X^c = X_1^p \cup X_2^p \cup \cdots \cup X_n^p,$$
$$\text{и } \Gamma^c = \Gamma_1^p \cup \Gamma_2^p \cup \cdots \Gamma_n^p.$$

Данные определения модуля, сегмента, программы и комплекса не носят абсолютный характер, а введены для обозначения объектов, используемых на этапе комплексирования. Поэтому эти понятия могут отличаться от аналогичных, рассматриваемых в других контекстах.

4.3. МАТРИЧНОЕ ПРЕДСТАВЛЕНИЕ ГРАФОВ МОДУЛЬНЫХ СТРУКТУР

Для определения основных операций над модульными структурами используем матричное представление их графов. Матричные представления часто используются в различных работах. Например, матрицы вызовов и матрицы достижимости в [125, 134, 152, 209] эквивалентны матрицам смежности ориентированных графов.

В настоящей работе в качестве матричного представления используется матрица вызовов. Элемент матрицы m_{ij} определяет количество обращений (операторов вызова) из модуля, соответствующего индексу i, к модулю, соответствующему индексу j. Кроме матрицы вызовов для дальнейшего анализа понадобится характеристический вектор, для каждой компоненты i которого $V_i = \chi(x_i)$. Для графа модульной структуры, приведенной на рис. 4.1, характеристический вектор и матрица вызовов будут иметь следующий вид:

$$V = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad M = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.2}$$

Проведем анализ матриц вызовов и характеристических векторов для графов модульных структур, соответствующих различным типам программных агрегатов.

Для **модулей** с графами, представленными на рис. 4.2, векторы и матрицы имеют следующий вид:

$$V_5^m = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad M_5^m = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; \quad V_6^m = (1), \quad M_6^m = (0). \tag{4.3}$$

Только один элемент характеристического вектора равен единице, и только в одной строке матрицы могут находиться ненулевые элементы.

Для **сегментов** с графами, представленными на рис. 4.3, векторы и матрицы записываются в таком виде:

$$V_3^s = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad M_3^s = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; \quad V_1^s = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

$$M_1^s = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad V_5^s = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad M_5^s = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{4.4}$$

Для **программы** с графом, представленным на рис. 4.1, характеристический вектор и матрица вызовов совпадают с V и M соответственно и определяются (4.2). Все элементы V равны единице.

Для **комплекса** характеристический вектор и матрица вызовов имеют следующий вид:

$$V^c = \begin{pmatrix} V_1^p \\ V_2^p \\ \vdots \\ V_n^p \end{pmatrix}, \quad M^c = \left\langle \begin{array}{c|c|c|c} M_1^p & 0 & \cdots & 0 \\ \hline 0 & M_2^p & & 0 \\ \hline \vdots & & & \\ \hline 0 & 0 & \cdots & M_n^p \end{array} \right\rangle. \tag{4.5}$$

Здесь V_i^p и M_i^p $(i = \overline{1, n})$ обозначают характеристический вектор и матрицу вызовов для графа i-й программы, входящей в комплекс.

В дальнейшем матричное представление используется для анализа операций над модульными структурами.

4.4. ОТНОШЕНИЕ ДОСТИЖИМОСТИ ДЛЯ ГРАФОВ МОДУЛЬНЫХ СТРУКТУР

Пусть $G = (X, \Gamma)$ — граф модульной структуры, x_i, x_j — вершины, принадлежащие X. Если в графе G существует ориентированная цепь от x_i к x_j, то вершина x_j достижима из вершины x_i. Справедливо следующее утверждение: если вершина x_j достижима из x_i, а x_l — из x_j, то x_l достижима из x_i. Доказательство этого факта очевидно.

Рассмотрим бинарное отношение на множестве X, которое определяет достижимость между вершинами графа. Введем обозначение $x_i \to x_j$ для достижимости вершины x_j из x_i. Отношение транзитивно. Обозначим через $D(x_i)$ множество вершин графа G, достижимых из x_i. Тогда равенство

$$\bar{x}_i = \{x_i\} \cup D(x_i) \tag{4.6}$$

определяет транзитивное замыкание для x_i по отношению достижимости. Докажем следующую теорему.

Теорема 4.1. *Для выбранной компоненты связности графа модульной структуры любая вершина достижима из корневой, соответствующей данной компоненте, т. е. выполняется равенство* (x_1 — *корневая вершина*)

$$\bar{x}_1 = \{x_1\} \cup D(x_1) = X. \tag{4.7}$$

Доказательство. Предположим, вершина x_i $(x_i \in X)$ недостижима из x_1. Тогда $x_i \notin \bar{x}_1$ и множество $X' = X \setminus \bar{x}_1$ непусто. Поскольку выбранная компонента графа связна, то существуют вершина $x_j \in \bar{x}_1$ и цепь $H(x_i, x_j)$, ведущая от x_i к x_j. Исходя из ациклличности графа G, в X' должна существовать простая цепь $H(x_l, x_i)$, где в вершину x_l не входят дуги (данная цепь может быть пустой, если X' состоит только из x_i). Рассмотрим цепь $H(x_l, x_j) = H(x_l, x_i) \cup$

$\bigcup H(x_l, x_j)$. Это означает, что модуль x_j достижим из вершин x_1 и x_l и обе вершины не содержат входящих дуг. А это противоречит определению графа модульной структуры с единственной корневой вершиной. Теорема доказана.

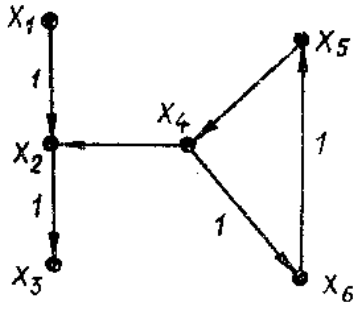


Рис. 4.4. Граф, содержащий ориентированный цикл

Результаты данной теоремы важны для обоснования требования отсутствия ориентированных циклов в графе модульной структуры относительно понятия достижимости. Рассмотрим граф, приведенный на рис. 4.4.

Из этого рисунка ясно, что модули, соответствующие вершинам x_4, x_5, x_6, никогда выполняться не будут. Таким образом, результаты теоремы 4.1 усиливают требование отсутствия ориентированных циклов в графе модульной структуры.

Проведем анализ матричного представления отношения достижимости для графов модульной структуры. В качестве примера рассмотрим граф, приведенный на рис. 4.1. Его матрица достижимости

$$A = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{matrix} \\ \left\| \begin{matrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right\| \end{array}. \qquad (4.8)$$

Коэффициент $a_{ij} = 1$, если модуль, соответствующий индексу l, достижим из модуля, соответствующего индексу i. Следующие результаты основаны на известной теореме из теории графов.

Теорема 4.2. *Коэффициент m_{ij} l-й степени матрицы смежности M^l определяет количество различных маршрутов, содержащих l дуг и связывающих вершину x_i с вершиной x_j ориентированного графа.*

Доказательство этой теоремы приводится в [99]. Рассмотрим следующие три следствия из этой теоремы.

Следствие 4.2.1. Матрица $\overline{M} = \sum_{l=1}^{n} M^l$, где M — матрица смежности ориентированного графа с n вершинами, совпадает с точностью до числовых значений коэффициентов с матрицей достижимости A.

Доказательство. В ориентированном графе, содержащем n вершин, максимальная длина пути без повторяющихся дуг не может превышать n. Поэтому последовательность степеней матрицы смежности M^l, где $l = 1, 2, \ldots, n$, определяет количество всех возможных путей в графе с количеством дуг $\leqslant n$. Пусть коэффициент $\overline{m}_{ij}$ матрицы $\overline{M}$ отличен от нуля. Это означает, что существует степень матрицы M^l, у которой соответствующий коэффициент m_{ij} также отличен от нуля. Следовательно, существует путь, идущий от вершины x_i к

Пусть $S_1 = (G_1, \chi_1)$ и $S_2 = (G_2, \chi_2)$ — две модульные структуры с графами $G_1 = (X_1, \Gamma_1)$ и $G_2 = (X_2, \Gamma_2)$ соответственно. Введем следующие обозначения:

$D(x)$ — множество вершин, достижимых из вершины x;

$D^*(x)$ — множество вершин, из которых достижима вершина x.

Для одинаковых вершин, входящих в графы G_1 и G_2, будут использоваться одинаковые обозначения.

Рассмотрим основные операции над модульными структурами. Операция *объединения*

$$S = S_1 \cup S_2 \tag{4.9}$$

предназначена для формирования модульной структуры комплекса и формально определяется следующим образом (S_1 и S_2 — любые модульные структуры, удовлетворяющие определению в п. 4.1):

$$G = G_1 \oplus G_2, \quad X = X_1 \oplus X_2, \quad \Gamma = \Gamma_1 \oplus \Gamma_2, \tag{4.10}$$

где символ $\oplus$ обозначает прямую сумму, и

$\chi(x) = \chi_1(x)$, если $x \in X_1$;

$\chi(x) = \chi_2(x)$, если $x \in X_2$.

Одинаковые вершины, входящие в G_1 и G_2, операцией объединения модульных структур рассматриваются как разные объекты. Поэтому характеристический вектор и матрица вызовов для модульной структуры S определяются так:

$$V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}, \quad M = \left(\begin{array}{c|c} M_1 & 0 \\ \hline 0 & M_2 \end{array}\right), \tag{4.11}$$

где $V_{1,2}$ и $M_{1,2}$ — характеристические векторы и матрицы вызовов для модульных структур S_1 и S_2 соответственно. Операция объединения ассоциативна, но не коммутативна — порядок следования операндов определяет порядок выполнения компонент комплекса. Необходимо отметить, что если операнды S_1 и S_2 удовлетворяют условиям определения модульных структур, то результат S также будет удовлетворять тем же требованиям.

Операция объединения модульных структур увеличивает число компонент связности соответствующего графа. Кроме того, графы структур, соответствующие операндам, сами могут иметь несколько компонент связности. Для остальных операций графы модульных структур операндов и результата имеют единственную компоненту связности.

Рассмотрим операцию *соединения*. Через x_i и x_j обозначим корневые вершины для графов G_1 и G_2 модульных структур S_1 и S_2 соответственно. Если данные структуры удовлетворяют условиям:

множество $X' = X_1 \cap X_2$ непусто;

вершина $x_j \in X'$ и $\chi(x_j) = 0$;

$D^*(x) \cap D(x) = 0$ для каждого $x \in X'$, где $D^*(x) \subset X_1$ и $D(x) \subset X_2$,

то операция соединения, обозначаемая

$$S = S_1 + S_2, \tag{4.12}$$

определяется следующим образом:

$$G = G_1 \cup G_2, \quad X = X_1 \cup X_2, \quad \Gamma = \Gamma_1 \cup \Gamma_2, \tag{4.13}$$

и для характеристической функции χ выполняется:

$$\chi(x) = \chi_1(x), \text{ если } x \in X_1 \setminus X';$$
$$\chi(x) = \max(\chi_1(x), \chi_2(x)), \text{ если } x \in X';$$
$$\chi(x) = \chi_2(x), \text{ если } x \in X_2 \setminus X'.$$

Первое условие означает, что в графах G_1 и G_2 имеются общие вершины. Согласно второму условию корневая вершина G_2 принадлежит общей части и для S_1 объект, соответствующий x_j, еще не включен в модульную структуру. Третье условие запрещает существование циклов в графе результата. Действительно, если существует $x_n \in \in D^*(x) \cap D(x)$, то $x_n > x$ и $x > x_n$, что означает существование цикла. Так как не все S_1 и S_2 удовлетворяют приведенным выше условиям, то операция частичная.

Определим принадлежность результата операции соединения к классу рассматриваемых модульных структур. Поскольку X' непусто, то граф G имеет одну компоненту связности. Корневой вершиной графа G является x_i. Сам граф G не имеет ориентированных циклов, т. е. ацикличен. Таким образом, S принадлежит к классу рассматриваемых модульных структур.

Операция соединения не коммутативная и в общем случае не ассоциативна. Чтобы показать последний факт, рассмотрим результат $S = (S_1 + S_2) + S_3$, где корневые вершины графов G_2 и G_3 входят в состав вершин графа G_1 и $X_2 \cap X_3 \neq \varnothing$. Тогда результат $S_2 + S_3$ не определен.

Рассмотрим операцию **проекции**. Пусть $S_1 = (G_1, \chi_1)$ — модульная структура и $x_i \in X_1$. Операция проекции модульной структуры на вершину графа G_1, обозначаемая как $S = Pr_{xi}(S_1)$, определяется следующим образом:

$$G(X, \Gamma), \quad X = \overline{x}_i, \quad \Gamma = \{(x_l, x_j, K) \mid x_l, x_j \in X\}, \tag{4.14}$$

и для характеристической функции

$\chi(x) = \chi_1(x)$, если $x \in X$.

Операция проекции определяет из модульной структуры S_1 подструктуру S. Проверим принадлежность S классу рассматриваемых модульных структур. Если граф модульной структуры S_1 связан и ацикличен, то теми же свойствами будет обладать и граф S. Существует единственная корневая вершина x_i в графе G. Таким образом, модульная структура S принадлежит рассматриваемому классу.

Операция **разности** для модульных структур определяется следующим образом. Пусть $S_1 = (G_1, \chi_1)$ — модульная структура и $x_i \in X_1$. Операция разности выполняется между модульной структурой и ее проекцией на вершину x_i соответствующего графа (x_i не является корневой вершиной графа G_1). Формально операция разности модульной структуры

$$S = S_1 - Pr_{xi}(S_1) \tag{4.15}$$

определяется следующим образом:

$$G = (X,\ \Gamma),\quad X = (X_1 \setminus \bar{x}_i) \cup X', \tag{4.16}$$
$$\Gamma = \{(x_l,\ x_j,\ K) \mid x_l,\ x_j \in X\},$$

где множество X' состоит из таких элементов, для которых

$$X' = \{x'_j \mid (x_l \in X_1 \setminus x_i) \,\&\, (x'_j \in \bar{x}_i) \,\&\, (x_l,\ x'_j,\ K) \in \Gamma\}. \tag{4.17}$$

Характеристическая функция χ определяется так:

$$\chi(x) = \chi_1(x),\ \text{если}\ x \in X_1 \setminus \bar{x}_i;$$
$$\chi(x) = 0,\ \text{если}\ x \in X'.$$

Во множество X включаются вершины, которые не вошли во множество $\bar{x}_i$, и те из вершин $\bar{x}_i$, в которые входят дуги из вершины $X_1 \setminus \bar{x}_i$ (множество X'). Характеристическая функция для элементов $x' \in X'$ равна нулю. Операция разности модульных структур определена так, чтобы быть обратной к операции соединения, т. е. чтобы выполнялось равенство

$$S - P_{r_{i\subseteq}}(S) + P_{r_{i\subseteq}}(S) = S. \tag{4.18}$$

Проверим принадлежность S, определяемой в (4.15), к классу рассматриваемых модульных структур. Если граф G_1 связан и ациклличен, то этими же свойствами будет обладать граф G. Корневая вершина G совпадает с корневой вершиной G_1. Таким образом, S удовлетворяет условиям определения модульной структуры, приведенным в п. 4.1.

Пусть S^* обозначает множество модульных структур, заданное на прямом произведении $G^* \times \chi^*$, где G^* и χ^* — соответственно множество графов и характеристических функций. Обозначим через Ω множество введенных операций над модульными структурами и предикаты C и R, рассмотренные в п. 4.2:

$$\Omega = \{\cup,\ +,\ -,\ P_r,\ C,\ R\}. \tag{4.19}$$

Этим мы определяем частичную алгебраическую систему $\mathfrak{U} = \langle S^*,\ \Omega\rangle$ над множеством модульных структур. Согласно определению типов программных агрегатов и операций над модульными структурами необходимо отметить следующее:

операция объединения применяется для программ и комплексов, результатом является комплекс;

операция соединения применяется для модулей и сегментов, результатом могут быть сегмент или программа;

операция проекции используется для программ и сегментов, результатом является модуль или сегмент;

операция разности применяется для программ и сегментов, результатом может быть модуль или сегмент.

4.6. ИСПОЛЬЗОВАНИЕ ОПЕРАЦИЙ ДЛЯ ПОСТРОЕНИЯ МОДУЛЬНЫХ СТРУКТУР

Рассмотрим основные задачи построения программных агрегатов и алгоритмы их решения. Они взяты из практики комплексирования программных средств, и этим во многом определилась реализация алгоритмов решения этих задач.

Задача 1. Дано множество модулей, входящих в состав программы, и имя главного (корневого) модуля. Построить модульную структуру программы.

Задача 2. Дано множество модулей, входящих в состав программы. Имя главного модуля неизвестно. Построить модульную структуру.

Задача 3. Дано множество модулей, реализующих некоторые функции предметной области, и имя главного модуля для одной из программ. Построить модульную структуру программы.

Задача 4. Дано множество модулей реализации функции предметной области и последовательность имен главных модулей нескольких программ. Построить модульную структуру для комплекса программ.

Задача 5. Заменить в модульной структуре один или несколько модулей новыми.

Задача 6. Выделить из модульной структуры объекты и включить их в другую структуру. В этой задаче под объектами понимается любая часть модульной структуры, т. е. любой подграф ее графа.

При решении данных задач будут использоваться введенные операции над модульными структурами. Для практического применения используется матричное представление объектов, а в качестве операций рассматриваются преобразования соответствующих матриц. Так, множества, характеристические функции представляются векторами, графы — матрицами вызовов и т. д. Аналогичные представления применяются и для объектов, используемых в алгоритмах решения задач. Технические аспекты реализации матричных представлений объектов и операций принципиальных трудностей не представляют и в данной работе не рассматриваются.

Рассмотрим решение каждой из поставленных задач, используя операции над модульными структурами.

1. Пусть X^p — множество вершин графа, соответствующее множеству модулей программы. Упорядочим его так, чтобы для каждых $x_i, x_j \in X^p$ из условия, что модуль, соответствующий x_i, вызывает модуль, соответствующий x_j, следует, что $j > i$. Если X^p таким образом не удается упорядочить, то в графе модульной структуры возникнут циклы, что противоречит необходимости ацикличности графа.

Обозначим вершину, соответствующую главному модулю, через x_1. Основной метод решения данной задача состоит в постепенном наращивании модульной структуры «сверху-вниз». С каждым модулем, соответствующим вершине x_i, связана модульная структура $S_i^m = (G_i^m, \chi_i^m)$, где $G_i^m = (\chi_i^m, \Gamma^m)$. Множество X_i^m включает x_i и вершины, соответствующие модулям, к которым имеются обращения из данного

модуля. Множество Γ_i^m соответствует множеству вызовов из модуля, соответствующего x_i. Характеристическая функция для x_i равна единице и нулю — для остальных вершин. Если из модуля, соответствующего x_i, нет обращения к другим модулям, то $X_i^m = \{x_i\}$, $\Gamma_i^m = 0$, $\chi(x_i) = 1$. Пусть Y — множество вершин графа, для которых $\chi = 0$.

Шаг 1. Вводим начальные значения: $S = S_1^m$; $Y := X_1^m \setminus \{x_1\}$.

Шаг 2. Если $C(S) = 1$, то перейти на шаг 6.

Шаг 3. Выберем первый элемент $x_i \in Y$. Множество Y непусто и конечно.

Шаг 4. $S := S + S_i^m$, $Y := (Y \setminus \{x_i\}) \cup (x_i^m \setminus \{x_i\})$. На данном шаге происходит наращивание модульной структуры и изменение множества Y.

Шаг 5. Переход на шаг 2.

Шаг 6. Выход.

Докажем корректность и сходимость данного алгоритма. На шаге 4 выполняется операция соединения структур. Данная операция корректна, так как ее условия выполняются:

$$x_i \in X \cap X_i^m,$$
$$\chi(x_i) = 0 \text{ и } \chi_i^m(x_i) = 1,$$

циклы отсутствуют согласно упорядоченности множества X^p.

Рассмотрим изменение множества Y на шаге 4. Согласно результатам п. 4.4, в графе модульной структуры обязательно существуют вершины, из которых исходят дуги. Исходя из этого факта и конечности множества X^p, следует, что существует x_i такое, что $Y \setminus \setminus \{x_i\} = 0$ и $x_i^m \setminus \{x_i\} = 0$. В результате условие шага 2 истинно и осуществляется переход на шаг 6 — завершение работы алгоритма.

Необходимо отметить, что данный алгоритм имеет практическое воплощение в процессе комплексирования, основанном на применении специальной компоненты операционной системы — редактора связей, построителя задач, комплексатора и т. д.

2. Для решения второй задачи достаточно упорядочить множество X^p, как и для условия первой. Тогда первый элемент X^p соответствует главному модулю и можно воспользоваться алгоритмом решения первой задачи.

3. Для решения третьей задачи проведем аналогичное упорядочение модулей предметной области. Единственное условие состоит в том, чтобы $x_1 \in X$ соответствовал главный модуль программы. Тогда можно воспользоваться алгоритмом решения первой задачи.

4. Пусть X^c — множество вершин, соответствующих модулям реализации функции предметной области, и $x' = \{x_1, x_2, \ldots, x_n\}$ — множество вершин, соответствующих главным модулям программ ($X' \subset X^c$). Алгоритм решения следующий.

Шаг 1. Начальное значение: $i := 1$, $S := 0$.

Шаг 2. Выбрать $x_i \in X'$ и построить модульную структуру с главным модулем, соответствующим корневой вершине x_i. Построе-

ние выполняется согласно алгоритму третьей задачи. Пусть S_i^p — соответствующая модульная структура.

Шаг 3. $S := S \cup S_i^p$. На данном шаге используется операция объединения модульных структур.

Шаг 4. $i := i + 1$. Если $i \leqslant n$, то переход на шаг 2.

Шаг 5. Выход.

Корректность и сходимость данного алгоритма очевидны и не требуют доказательства.

5. Пусть $S = (G, \chi)$ — модульная структура с графом $G = (X, \Gamma)$. Необходимо заменить модуль программы, соответствующий вершине $x_i \in X$, новым модулем x_i'. Сложность данной задачи состоит в том, что с заменой модуля может измениться модульная структура S, так как x_i в общем случае является корневой вершиной для некоторого графа. Данный алгоритм будет иметь следующий вид.

Шаг 1. Построить модульную структуру S_i' для корневой вершины x_i' согласно алгоритмам задач 1 — 3 (предполагается, что множество модулей для данной структуры известно до ее построения).

Шаг 2. $S := S - P_{rx_i}(S)$. Исключается соответствующая часть модульной структуры S.

Шаг 3. Переобозначим x_i' через x_i, а S_i' — через S_i. Выполним операцию: $S = S + S_i$.

Шаг 4. В результате предыдущей операции в графе G модульной структуры могут существовать вершины с $\chi = 0$, т. е. $C(S) = 0$. Тогда необходимо применить алгоритм первой задачи, полагая в качестве множества Y множество таких вершин.

Выполнение операции соединения на шаге 3 должно удовлетворять необходимым условиям.

6. Данная задача подобна задаче 5. Пусть $S_1 = (G_1, \chi_1)$ — исходная модульная структура с графом $G_1 = (X_1, \Gamma_1)$ и $x_i \in X_1$. Граф $G_2 = (X_2, \Gamma_2)$ модульной структуры $S_2 = (G_2, \chi_2)$ имеет вершину x_i с $\chi(x_i) = 0$. Тогда алгоритм решения данной задачи следующий.

Шаг 1. $S_i = P_{rx_i}(S_1)$.

Шаг 2. $S := S_2 + S_i$. Операция соединения модульных структур должна удовлетворять необходимым условиям.

Шаг 3. Если $C(S) = 0$, то выполнять действия, аналогичные шагу 4 пятой задачи.

Как видно из приведенных выше алгоритмов, для решения задач 1—6 использовались все четыре операции над модульными структурами.

4.7. ТИПЫ И МЕТОДЫ ПОСТРОЕНИЯ ПРОГРАММНЫХ СТРУКТУР

Среди многообразия программных структур выделяются три основные — простая, оверлейная (структура с перекрытием) и динамическая. Основное назначение различных структур — наиболее оптимальное использование основной памяти во время выполнения программного агрегата. Поэтому тип программной структуры относится

к динамическим характеристикам соответствующей модульной структуры.

Простая структура. Программный агрегат с простой структурой создается на этапе комплексирования, где все связи между модулями фиксированы и не меняются во время выполнения. Объем основной памяти, занимаемой программным агрегатом с данным типом структуры, постоянен и равен сумме объемов отдельных модулей: $V_s = \sum_{i=1}^{n} v_i$, где v_i — объем памяти, занимаемый i-м модулем. Соответствующий граф модульной структуры всегда связан.

Оверлейная структура. Программный агрегат с оверлейной структурой также создается на этапе комплексирования. Однако связи между модулями не такие жесткие, и их последовательность определяется относительно входящих в одну цепочку модулей. Модули на этапе выполнения загружаются в основную память в момент обращения к ним. После завершения работы память освобождается и может быть использована для загрузки другого модуля.

Компонента программного агрегата, загружаемая при однократном обращении, называется оверлейным сегментом. Его граф соответствует определению графа модульной структуры для сегмента. В один оверлейный сегмент включаются модули, между которыми существуют частые обращения, что экономит время выполнения программного агрегата за счет уменьшения количества операций ввода с устройств внешней памяти.

Как и для случая простой структуры, граф соответствующей модульной структуры также связный, и матрица вызовов имеет обычный вид, например (4.2). Объем требуемой основной памяти зависит от количества и состава оверлейных сегментов. Максимальный объем памяти равен сумме объемов памяти отдельных модулей: $v_0^{\max} = V_s = \sum_{i=1}^{n} v_i$. Минимальный объем памяти, требуемый при выполнении, равен максимальному объему памяти среди всех возможных цепочек программного агрегата. Определим этот объем, используя модифицированный алгоритм Флойда [136]. Сам алгоритм Флойда рассчитан на определение кратчайшего пути в графе, где каждой дуге соответствует весовой коэффициент, называемый длиной дуги. В нашем случае коэффициенты имеют вершины графа модульной структуры. Для применения алгоритма Флойда выполним следующие преобразования.

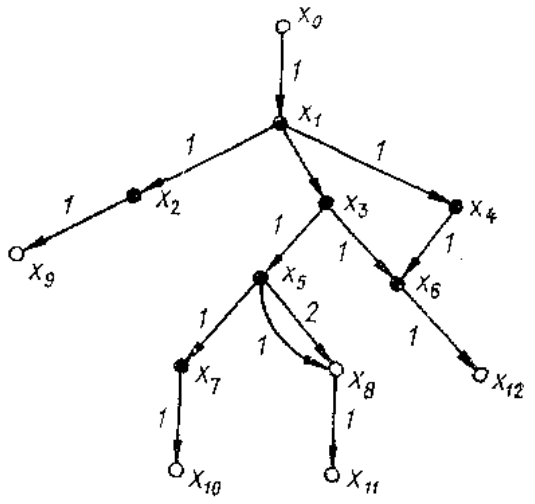


Рис. 4.5. Граф модифицированной модульной структуры

1. Дополним граф модульной структуры новыми вершинами и дугами. Вершинами являются x_0, x_{n+1}, x_{n+2}, ..., x_{n+m}, где m —количество концевых вершин. Новые дуги—$(x_0, x_1, 1)$,

$(x_{r1}, x_{n+1}, 1), \ldots, (x_{rn}, x_{n+m}, 1)$ где x_1 соответствует главному модулю, а все x_i — концевые вершины. После выполнения таких операций граф модульной структуры, представленной на рис. 4.1, приведен на рис. 4.5. Новыми вершинами являются x_0, x_9, x_{10}, x_{11}, x_{12}. Всем им ставим в соответствие весовые коэффициенты, равные нулю:

$$v_0 = v_9 = v_{10} = v_{11} = v_{12}.$$

2. Каждой дуге вида (x_i, x_j, k) ставим в соответствие коэффициент $v_{ij} = \frac{v_i + v_j}{2}$. Рассмотрим все маршруты, ведущие от x_0 к одной из остальных дополнительных вершин. Длина маршрута

$$l_{0,\,n+p} = v_{01} + \cdots + v_{rp,\,n+p} = \frac{v_0 + v_1}{2} + \cdots + \frac{v_{2p} + v_{n+p}}{2} =$$

$$= \frac{v_0}{2} + v_1 + \cdots + v_{rp} + \frac{v_{n+p}}{2} = v_1 + \cdots + v_{rp}.$$

Длина маршрута $l_{0,\,n+p}$ будет равна сумме объемов памяти модулей для пути $x_1, \ldots, x_{r_p}$. Таким образом, применяя алгоритм Флойда к графу, изображенному на рис. 4.2, мы решаем задачу вычисления объема памяти для максимальной цепочки.

3. Матрицу вызовов заменим матрицей путей. Для каждого $m_{ij} > 0$ на соответствующем месте будет находиться значение v_{ij}. Значения $m_{ij} = 0$ заменяются на $-\infty$. Программа, реализующая алгоритм Флойда, будет иметь следующий вид (предполагается, что матрица путей описана как двухмерная матрица размером $n \times n$):

```
for k = 1 to n do
for i = 1 to n do
for j = 1 to n do
if M [i, j] < M [i, k] + M [k, j] then
  M [i, j]: = M [i, k] + M [k, j].
```

В результате работы этого алгоритма будет построена матрица максимальных путей. Максимальное из значений $l_{0,\,n+p}$ будет определять необходимый минимальный объем памяти $v_0^{\min}$ для программного агрегата с оверлейной структурой. Саму оверлейную структуру для значений $V_0^{\min} \leqslant V_0 \leqslant V_0^{\max}$ можно построить, следуя алгоритмам, предложенным в [101, 102].

Качественно зависимость V_0 от числа оверлейных сегментов представлена на рис. 4.6. Здесь n — число модулей в программном агрегате. Несмотря на различный вид кривых, они имеют общую закономерность — любое V_0 заключено между значениями $v_0^{\max}$ и $v_0^{\min}$.

Динамическая структура. Механизм динамической связи между модулями отличен от механизма обращения с помощью оператора вызова CALL. Поэтому для описания графа модульной структуры необходимы дополнительные средства.

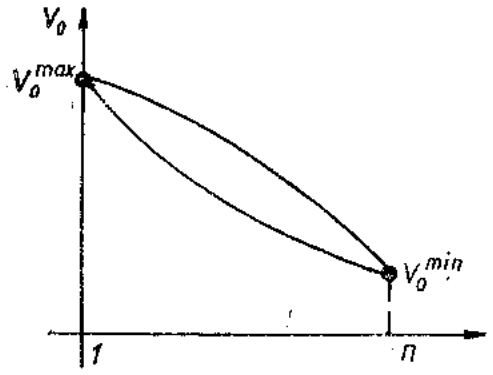
Как и для оверлейной структуры, загрузка в основную память динамических объектов происходит при обращении к ним. По анало-

гии назовем объем, загружаемый при однократном обращении, динамическим сегментом. Каждый динамический сегмент имеет свою собственную модульную структуру, для которой составляется матрица вызовов. Если в разных динамических сегментах встречаются одинаковые модули, то они являются разными объектами в графе модульной структуры. Для иллюстрации используем исходный граф (см. рис. 4.1). Пусть из модуля, соответствующего вершине x_1, динамически вызывается модуль, соответствующий вершине x_3. Полученный измененный граф приведен на рис. 4.7. Пунктирная стрелка обозначает динамический вызов. Модуль, соответствующий вершине x_6, встречается дважды.

Построим матрицу вызовов для данного программного агрегата. Каждому динамическому сегменту будет соответствовать своя клетка. Чтобы отличить динамический вызов от вызовов по оператору CALL, соответствующие элементы матрицы будут содержать отрицательные числа, абсолютные значения которых будут определять количество динамических вызовов между данными парами модулей. Матрица вызовов будет иметь вид

$$M = \begin{array}{|ccccccccc|} x_1 & x_2 & x_4 & x_6 & x_3 & x_5 & x_6' & x_7 & x_8 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}. \qquad (4.20)$$

Исследуем качественную зависимость необходимого объема оперативной памяти от количества динамических сегментов. При одном сегменте, т. е. программном агрегате простой структуры, $V_d^1 = V_s$. Если каждый динамический сегмент состоит из одного модуля, то, как и в



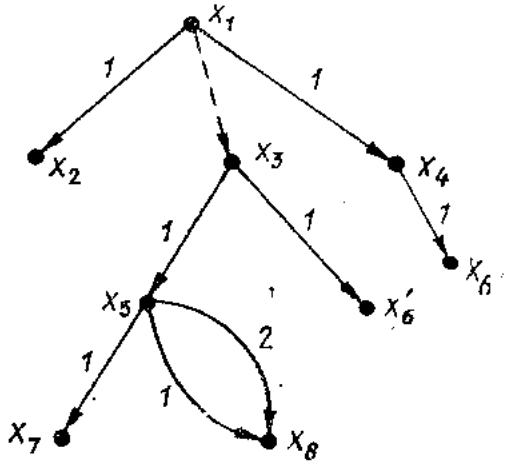


Рис. 4.6. Графики качественной зависимости V_0 от количества оверлейных сегментов

Рис. 4.7. Граф модульной структуры для динамического вызова

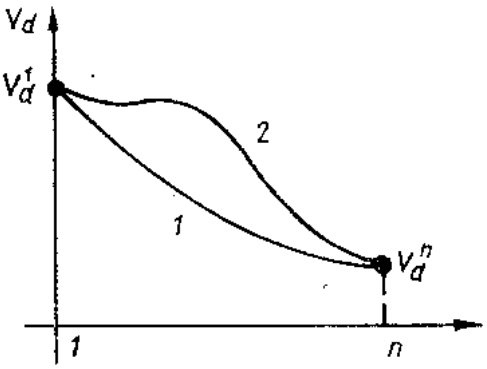


Рис. 4.8. Графики качественной зависимости V_α от количества динамических сегментов

случае оверлейной структуры, по модифицированному алгоритму Флойда находится максимальный путь и $V_d^n = V_0^{min}$. Для промежуточных значений зависимость имеет более сложный характер. На рис. 4.8 представлены две кривые. Здесь n — число модулей . в программном агрегате. Кривая *1* описывает зависимость, при которой в разных сегментах нет одинаковых модулей. Кривая *2* описывает зависимость для случая, когда у разных сегментов имеются одинаковые модули. Требуемая память увеличивается за счет дублирования таких модулей, как в рассмотренном выше примере. Однако вторая зависимость характерна и для случая, когда в динамических сегментах нет одинаковых модулей, но сами модули написаны на ЯП высокого уровня. Это вызвано тем, что в состав каждого динамического сегмента включаются одинаковые служебные программы — управления памятью, вводом-выводом, обработки аварийных ситуаций и т. д. За счет дублирования этих программ происходит увеличение необходимого объема основной памяти.

Таким образом, кривая *1* характерна только для программных агрегатов, состоящих из модулей, написанных на Ассемблере, и графов модульной структуры в виде дерева, что гарантирует отсутствие в разных динамических сегментах одинаковых модулей. Несмотря на явный недостаток динамической структуры в сравнении с оверлейной в экономии памяти, у нее есть существенное преимущество — независимость от редактирования связей. Каждый динамический сегмент может быть изменен, а редактирование связей для всего программного агрегата не требуется.

4.8. МЕТОД РЕАЛИЗАЦИИ СВЯЗИ ПАРЫ МОДУЛЕЙ В МОДУЛЬНОЙ СТРУКТУРЕ

Модульная структура, представленная графом G и определенная на множестве модулей $Y = \{y_1, y_2, \ldots, y_m\}$, описываемых в классе ЯП L (ПЛ/1, Фортран, Кобол, Ассемблер для ОС ЕС), служит основой для автоматизированного комплексирования по ней модулей в программный агрегат. При этом для каждой пары модулей y_i, y_j (i, j — языки из L), связанных на графе отношением вызова (типа CALL), формируется модуль связи y'_{ij}. В общем случае для простых структур программ он содержит операторы прямого и обратного преобразований типов данных, передаваемых от вызывающего модуля (в i-языке) вызываемому (в j-языке) и обратно.

Для построения программных структур, в графе которых вершины — модули отмечены еще и специальными символами ρ, указывающими на тип вызова (динамический, оверлейный, подзадачный), в модуле связи, кроме указанных операторов преобразования типов

данных, размещаются и операторы формирования среды выполнения (динамической, оверлейной и т. п.) соответствующей части агрегата.

Символ ρ может принимать следующие значения:

ρ = Ω, что означает формирование оверлейного сегмента, начиная с имени модуля, который этот символ помечает;

ρ = ⋇ — определяет начало динамического сегмента с вершины, помеченной в графе этим символом;

ρ = + — отмечает в графе G модуль, который должен быть сформулирован как главный модуль подзадачи;

ρ = / — означает включение отладочной среды.

Используя эти обозначения, граф, ранее приведенный на рис. 4.1, примет вид графа, изображенного на рис. 4.9. В нем:

для сегмента, заданного графом $\Gamma = \{(x_5, x_7, 1), (x_5, x_8, 1), (x_5, x_8, 2)\}$ и помеченного именем x_5 со знаком Ω, в модуле связи x'_{58} будет сформирован фрагмент из операторов, обеспечивающих оверлейный вызов;

для сегмента, заданного графом $\Gamma = \{(x_4, x_6, 1)\}$, в модуле связи x'_{46} будет сформирован фрагмент из операторов, обеспечивающих динамический вызов.

Таким образом, для пары модулей x_i, x_j создается модуль связи x_{ij} вида

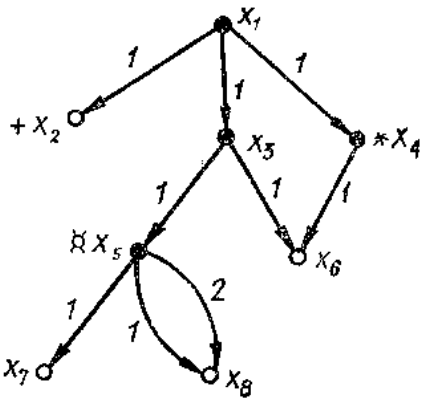
$$x'_{ij} = S_0 \divideontimes (S_1 \times S_1^T) \divideontimes (S_2 \times S_2^T) \divideontimes S_0^1,$$

где S_0 — фрагмент, задающий среду функционирования модуля x_j;

S_1 — фрагмент, включающий последовательность обращений к функциям из множества $\{P, C, S\}$, каждая из которых осуществляет необходимое преобразование фактических параметров при обращении на x_j-модуль;

S_1^T, $S_2^T \neq \varnothing$ означает наличие средств создания отладочной среды (по ρ = / в графе модульной программы) для x_i и x_j соответственно;

S_2 — фрагмент операторов по обратному преобразованию типов данных, передаваемых из x_j в x_i после его выполнения;



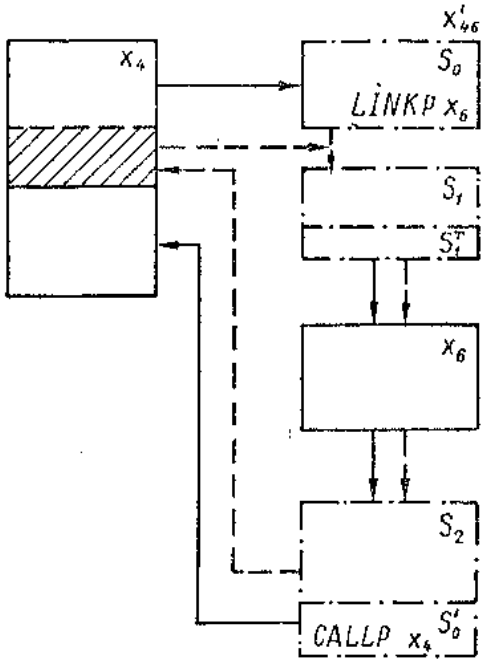
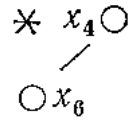


Рис. 4.9. Граф программного агрегата с управляющими отметками

Рис. 4.10. Структура программного агрегата

S_0^1 — фрагмент команд эпилога для восстановления среды модуля x_i.

Пример. Для пары модулей, заданных на графе (см. рис. 4.9)



структура соответствующей части программного агрегата, включая модуль связи, изображена на рис. 4.10. Аналогично реализуются связи разноязыковых модулей и для других видов вызова.

Автоматизация связи пары разноязыковых модулей в модульной структуре посредством модулей связи осуществляется на основе унификации объектов модульного типа и операций над этими объектами, определенных на множестве функций преобразования простых и структурных типов данных.

Унификация объектов определяется посредством задания для объектов описаний их паспортов.

Паспорт модуля — это специальный раздел в описании модуля, содержащий следующие виды информации:

имя паспорта, которое совпадает с именем модуля;

язык программирования данного модуля;

список формальных параметров модуля;

разделение формальных параметров на входные и выходные;

список вызываемых модулей;

список фактических параметров для каждого вызываемого модуля;

описание типов данных для всех параметров, указанных в паспорте;

описание файлов печати, используемых в модуле.

Каждый вид информации образует подраздел, не зависящий от ЯП, кроме подраздела описания типов данных, на котором описан модуль. Паспорт описывается на специальном языке комплексирования L', содержащем:

подмножества проекций языков из L, соответствующие описанию типов данных параметров, указанных в списках фактических и формальных параметров;

оператор описания графа и связи модулей в модульную структуру.

Язык L' позволяет описать паспорта модулей в классе языков L и графовую модель модульной структуры агрегата. При включении в класс L языков Модула-2, Си, Ада и др. в язык L' должны включаться средства конструирования новых типов данных, в том числе абстрактных типов данных, приведенные в [4, 110].

Модульная структура описывается в явном и неявном виде средствами языка L'. Неявное описание основано на том, что в паспорте модуля указываются имена вызываемых модулей. Это позволяет по паспорту главного (корневого) модуля построить для модульной структуры программный агрегат. Задачи построения агрегата модульных структур основаны на этом методе. Использование такого

подхода позволяет отождествить паспорт программного агрегата с паспортом его главного модуля.

Явное описание модульной структуры основано на машинном представлении графа в виде матрицы вызовов. Особенности задания матриц вызовов приведены выше.

В качестве агрегатов выступают: сегмент — SEG; программа — PROG; комплекс — COMP; пакет — PACT. Их паспорта формируются в процессе комплексирования и состоят из совокупности паспортов модулей, входящих в состав графа программного агрегата.

В языке конструирования имеются операторы, позволяющие описать указанные типы программных агрегатов с различными режимами их выполнения. Общая форма записи оператора связи имеет вид

LINK ⟨тип агрегата⟩ ⟨имя агрегата⟩ (⟨имя главного модуля⟩, ⟨дополнительный список имен модулей⟩) ⟨режим выполнения⟩.

Тип программного агрегата принимает следующие значения: SEG, PROG, COMP, PACT.

Имя агрегата соответствует имени, под которым программный агрегат после комплексирования, заносится в загрузочную библиотеку. Если создается сегмент, в дальнейшем включающийся в более сложный агрегат, то его имя должно совпадать с именем главного модуля. Для программы и комплекса это — уникальное имя, которое присваивается генерируемому корневому модулю.

Имя главного модуля модульной структуры указывается при комплексировании сегмента или программы. При построении комплекса этому имени соответствует имя первой выполняемой программы. Дополнительный список имен модулей указывается для построения в сегментах и программах оверлейной и динамической структур, а также для записи последовательности выполняемых программ комплекса.

Оператор позволяет записывать вид программных структур, отличных от простой. При этом знак ρ, заданный в вершинах графа, ставится перед именем модуля, что определяет тип вызова для данного модуля.

Режим выполнения определяется символом, принимающим следующие значения:

— построение графа модульной структуры и вывод на экран терминала;

0 — включение в модули-связки средств отладки модульной структуры.

Если символ режима выполнения отсутствует, то предполагается обычный режим.

Средства управления процессом комплексирования. Данные средства обеспечивают выполнение нескольких этапов построения модульных структур программ.

1. Ввод задания в языке *L'* на комплексирование программного агрегата и на выполнение синтаксического контроля операторов, содержащихся в задании.

2. Построение модульной структуры согласно используемым алгоритмам и операциям, описанным выше. На данном этапе проверяется наличие всех необходимых модулей, определяется их местона-

хождение (личная библиотека, банк модулей общего пользования и т. д.) и формируется матрица вызовов графа модульной структуры.

3. Генерация модулей связи для:

обеспечения связи пар разноязыковых модулей;

построения динамической, оверлейной или подзадачной структуры;

формирования отладочной среды программного агрегата;

генерации корневых модулей для программ и комплексов.

4. Трансляция модулей связи и модулей в языке L', входящих в состав программного агрегата. На данном этапе выполняется:

трансляция сгенерированных модулей связи;

трансляция исходных модулей в ЯП для данного агрегата, если ранее эта операция к ним не применялась;

подготовка всех модулей для выполнения процесса редактирования связей в модульной структуре для представления агрегата в загрузочном виде.

Данный этап не имеет непосредственного отношения к комплексированию, но для автоматизации этого процесса он необходим.

5. Организация построения различных программных структур путем генерации управляющих предложений для различных структур программ и выполнения редактирования связей для отдельных сегментов. Обычно в качестве редактора связей используется соответствующая программа операционной системы.

Практическая реализация методов управления модульными структурами нашла воплощение в средствах комплексирования системы АПРОП. Данные средства предназначены для:

представления модульных структур, в том числе матричной;

реализации алгебраической системы с операциями над множеством модульных структур;

построения различных типов программных агрегатов и программных структур;

отладки модульных структур.

4.9. МЕТОДЫ ОТЛАДКИ МОДУЛЬНЫХ СТРУКТУР

Применение данных методов предполагает, что отдельные модули входящие в модульную структуру, прошли этап автономной отладки. На средства автономной отладки никаких ограничений не накладывается. Задачи отладки модульных структур состоят в проверке правильности построения модульной структуры и выполнения программного агрегата, соответствующего данной модульной структуре. Рассмотрим эти задачи более подробно.

Проверка правильности построения модульной структуры. Существует два способа проверки. Первый основан на анализе результатов процесса редактирования связей (сборки модулей), выполняемого специальной программой операционной системы. Данный способ позволяет выявить грубые ошибки — отсутствие модулей в модульной структуре, к которым есть обращения. Эти ошибки являются следствием реального отсутствия модулей или неверного имени в операторе вызова LINK.

Второй способ основан на анализе самой модульной структуры, который заключается в следующем:

1. Визуальный анализ графа модульной структуры. Проверка правильности построения непосредственно осуществляется самим разработчиком ПО. Отображение графа модульной структуры на экране терминала или вывод его на бумажный носитель.

2. Проведение анализа матриц, описывающих модульные структуры, основанного на результатах п. 4.4. Для проверки правильности построения модульных структур используются матрицы вызовов и достижимости. В результате проведенного анализа устанавливается существование циклов, число маршрутов и достижимость между каждой парой модулей (данная информация используется для определения количества маршрутов при тестировании).

Перейдем к рассмотрению второй основной задачи отладки модульных структур.

Проверка правильности выполнения модульной структуры. Решение данной задачи тесно связано с реализацией проблем межъязыкового интерфейса. Проверка правильности выполнения модульной структуры предполагает отслеживание в динамике последовательности передач управления и данных между взаимодействующими модулями. Такая возможность позволяет фиксировать цепочки выполняемых модулей и выполнять трассировку передаваемых данных. Результаты этих операций отображаются на экран терминала или печать. На основе их анализа определяются:

правильность последовательности вызовов модулей в соответствии с графом модульной структуры и в зависимости от входных данных;

правильность передаваемых данных между взаимодействующими модулями согласно описанию их типов в списке формальных и фактических параметров;

последний выполняемый модуль в момент аварийной ситуации при выполнении программного агрегата;

некоторые виды зацикливания в модулях при выполнении программных агрегатов.

Реализация этих функций возможна только при применении шестого метода комплексирования (см. гл. 3) — использовании промежуточных модулей связи, в которые включаются средства для отладки модульных структур агрегатов. С их помощью обеспечиваются: фиксация момента входа в вызываемый модуль; фиксация момента выхода из вызываемого модуля; отображение и хранение значений входных параметров согласно описаниям их типов данных при передаче управления вызываемому модулю; отображение и хранение значений выходных (и, возможно, входных) параметров согласно описаниям их типов данных при возврате управления вызывающему модулю.

Средства отладки могут включать и дополнительные возможности: изменение значений передаваемых параметров при выполнении программного агрегата с использованием режима диалога; отображение файлов печати (значений параметров, цепочек вызовов модулей и т. п.) на экран терминала после работы каждого модуля или агрегата в целом.

Глава 5

МЕТОДЫ ПОСТРОЕНИЯ ИНТЕГРИРОВАННЫХ КОМПЛЕКСОВ

5.1. ОСОБЕННОСТИ ИНТЕРФЕЙСОВ ДЛЯ ИНТЕГРИРОВАННЫХ КОМПЛЕКСОВ

Ранее введенное понятие программного интерфейса рассматривалось для пары взаимодействующих компонент. Это полностью справедливо при комплексировании модулей, где требуется обеспечить интерфейс между вызывающим и вызываемым модулями. При переходе к интегрированным комплексам понятие программного интерфейса нуждается в расширении. Для этого рассмотрим схему ИК, представленную на рис. 5.1. Результатом работы программы P_1 служит файл F_1, а программы P_2 — файл F_2. Оба файла являются входными данными для программы P_3. Возможны следующие последовательности выполняемых программ: $P_1 — P_2 — P_3$ и $P_2 — P_1 — P_3$. Предположим, что представления данных в файлах F_1 и F_2 требуется изменить для работы программы P_3. Это означает, что требуется преобразование данных и необходимы соответствующие интерфейсы.

Рассмотрим интерфейс, требуемый для работы программы P_3. Очевидно, что прежнее понятие программного интерфейса в данном случае неприменимо, поскольку нельзя выделить конкретную пару взаимодействующих модулей, связь между которыми охватывала бы все данные для программы P_3. Этот пример показывает, что для интегрированных комплексов требуется иное понятие интерфейса.

Рассмотрим множество данных интегрированного комплекса, определенное во второй главе формулой (2.6):

$$D = (\bigcup_{i=1}^{s} D^i) \cup D^c. \qquad (5.1)$$

Дополним множество данных i-й компоненты D^i данными, характеризующими среды языков программирования, систем программирования и т.д. для модулей, входящих в состав этой компоненты. Примером таких данных могут служить данные среды ЯП, о которых говорилось в гл. 3. Они обычно недоступны прикладному программисту и необходимы для реализации внутренних механизмов управления и функционирования отдельных модулей и всей компоненты в целом. Перечислим наиболее важные типы таких данных:

списки областей используемой и свободной памяти;

списки областей сохранения регистров;

стек, содержащий внутренние данные, адреса передачи и возврата управления для программ с модульной или блочной структурой;

блоки описания задач;

блоки описания файлов и устройств ввода-вывода;

признаки, определяющие необходимые действия при возникновении специальных условий, программных прерываний и аварийных ситуаций;

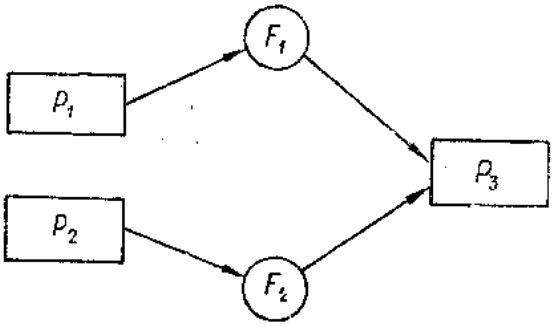


Рис. 5.1. Схема интегрированного комплекса

используемые шрифты для ввода информации с клавиатуры и вывода на терминал и печатающее устройство;

внутренние структуры для управления динамическими объектами, буферами ввода-вывода и т. д.

Внутренние структуры этих данных, их методы обработки зависят от конкретных операционных систем, трансляторов с ЯП, систем программирования и других инструментальных средств. Поэтому особенности их использования в настоящей книге не рассматриваются. Предполагается, что сами данные могут быть описаны в рамках теории структурной организации данных. Обычно на практике это условие выполняется.

Переопределив множества D^i, мы для упрощения изложения оставляем те же обозначения. Как было отмечено в гл. 2, каждое $d_j^i \in D^i$ характеризуется именем N_j^i, типом данных T_j^i и текущим значением V_j^i. Некоторые переменные, имеющие различные имена и принадлежащие различным компонентам, имеют одинаковое семантическое содержание, т. е. представляют различные реализации общей для нескольких компонент информационной структуры. Приводимые ниже примеры показывают наиболее типичные случаи данных с одинаковым семантическим содержанием.

1. На вход программы сортировки SORT поступает файл F_1, содержащий неупорядоченные записи. Результатом работы SORT служит упорядоченный файл F_2. Программа FORM формирует отчет на основании данных из файла F_3, содержащего упорядоченные записи. Если программа SORT будет выполнять сортировку записей со структурой, аналогичной структуре записи файла F_3, то обе программы могут быть объединены в единый программный комплекс. При этом файл F_2 и F_3 представляют собой данные с одинаковой семантикой. На практике соответствие между этими файлами может быть установлено следующими способами:

создание копии файла F_2 с именем F_3;

переименование имени F_2 в F_3 средствами ОС;

настройка программы FORM на обработку файла F_2, если имя определяется параметрически.

2. Программа FORM формирует отчет и выводит его в файл F_4. Программа REGISTR регистрирует полученные документы. Для

регистрации необходимы наименование отчета, дата и время его формирования. Эти данные программа REGISTER получает из командной строки (строки, содержащей имя вызываемой программы и передаваемые параметры). Если существуют универсальные средства интерфейса, позволяющие выбрать необходимую информацию из файла F_4 и сформулировать командную строку, то программы FORM и REGISTR можно объединить в единый программный комплекс. При этом поля из файла F_4, содержащие наименование отчета, дату, время, и соответствующие поля из командной строки будут представлять данные с одинаковым семантическим содержанием.

3. Некоторые операционные системы (например, MS DOS) позволяют в общесистемной области определять строковые константы с указанием символического имени для каждой строки. Строка может представлять имя файла, каталоги и т. д. Программы могут обращаться к этой области и по соответствующему символическому имени выбирать необходимую информацию. Может возникнуть ситуация, когда несколько программ должны получать одинаковую информацию из общесистемной области, но каждая программа содержит свое собственное символическое имя, определяющее данные с одинаковым семантическим содержанием. Реализация интерфейса для рассматриваемых программ может включать:

копирование информации в общесистемной области;

перед вызовом конкретной программы производить замену соответствующего символического имени.

Приведенные выше примеры не исчерпывают всех случаев данных с одинаковым семантическим содержанием. Их характерность состоит в различных комбинациях расположения данных: файл—файл, файл—оперативная память (ОП), ОП — ОП.

Рассмотрим множество D, определяемое в (5.1). Ему соответствует множество имен переменных N:

$$N = (\bigcup_{i=1}^{s} N^i) \cup N^c. \tag{5.2}$$

При этом положим, что все имена в $N^1, \ldots, N^s, N^c$ разные. Это условие не существенно, так как можно перейти к рассмотрению составных имен вида $P^i.N^i_j$, где P^i соответствует имени i-й программной компоненты, а $N^i_j \in N^i$. Эти имена явно отличаются.

Через E обозначим отношение на N, определяющее данные с одинаковым семантическим содержанием. Нетрудно заметить, что E — отношение эквивалентности. Оно определяет разбиение множества на классы эквивалентности N_p:

$$N = \bigcup_{p=1}^{r} \tilde{N}_p, \tag{5.3}$$

где $\tilde{N}_p \cap \tilde{N}_q = \varnothing$ при $p \neq q$. Через $\tilde{N}$ обозначим фактор-множество N/E по отношению эквивалентности E. Определим множество $G = \{G_1, \ldots, G_r\}$, где G_p — имя для класса эквивалентности $\tilde{N}_p$.

В качестве имени можно выбрать любое имя, являющееся представителем класса $\tilde{N}_p$. Однако для упрощения будем рассматривать имена G_p, отличные от имен из N.

Для каждой программы P^i множество данных D^i состоит из входных и выходных переменных. Перед выполнением P^i выходные переменные не определены. Для рассмотрения этой ситуации в нашей терминологии введем понятие «пустого значения» V_0^0, которое принимает любая неопределенная переменная независимо от ее типа. Соответственно введем в рассмотрение и понятие «пустого типа данных» T_0^0. Отметим, что переменная, имеющая тип T_0^0, всегда имеет значение V_0^0. Обратное выполняется не всегда — тип переменной может быть известен, но она сама не определена.

Рассмотрим множество $B = \{b_1, \ldots, b_r\}$, определяемое следующим образом. Каждый элемент b_p представляется тройкой (G_p, T_p, V_p). Для V_p имя G_p постоянно, а тип T_p и значение V_p меняются в процессе выполнения программ P^i, составляющих интегрированный комплекс. Перед началом выполнения $T_p = T_0^0$, $V_p = V_0^0$ при $p = \overline{1, r}$ (r — число программ в комплексе). В дальнейшем множество D будет называться базой данных, а B — информационной средой ИК. В каждый момент времени все T_p и V_p имеют конкретные значения (возможно, пустые). Набор конкретных значений для T_p и V_p будет определять состояние информационной среды ИК. В процессе выполнения программ состояние информационной среды $B = \{B^t\}$ меняется, т. е. существует функциональная зависимость вида $B^{t+1} = f(B^t, P^i)$, где B^{t+1} соответствует новому состоянию после выполнения программы P^i.

Рассмотрим более подробно переход от B^t к B^{t+1}. Программе P^i соответствует множество данных D^i. Пусть $d_j^i \in D$ и $d_j^i = (N_j^i, T_j^i, V_j^i)$. Имени N_j^i соответствует некоторый класс эквивалентности N_p, и, следовательно, $b_p = (G_p, T_p, V_p)$. Для входных переменных T_p и V_p должны быть определены, т. е. $T_p \neq T_0^0$, $V_p \neq V_0^0$. Выполнение этих условий рассматривалось в модели управления программными объектами (см. гл. 2). Пусть $T_p = T_l^k$ и $V_p = V_l^k$. Для выходных переменных из D^i соответствующие элементы b_p рассматриваются с $T_p = T_0^0$ и $V_p = V_0^0$.

Перед вызовом программы P^i данные $b_p = (G_p, T_l^k, V_l^k)$ должны быть преобразованы к представлению $d_j^i = (N_j^i, T_j^i, V_j^i)$. Эти преобразования являются частичными задачами в модели информационного сопряжения (см. гл. 2). Если существуют алгоритмы преобразований FT_{lj}^{ki} и FV_{lj}^{ki}, то происходит переход к промежуточному состоянию информационной среды $B^{t'}$. После выполнения программы P^i элементы b_p, соответствующие выходным данным из D^i, принимают вид $b_p = (G_p, T_j^i, V_j^i)$. Этим определяется новое состояние информационной среды b^{t+1}. При вызове следующей программы выполняются аналогичные действия.

Для завершения анализа необходимо рассмотреть начальное состояние информационной среды B^0. Ранее было отмечено, что к началу выполнения для всех $b_p = (G_p, T_p, V_p)$ $T_p = T_0^0$ и $V_p = V_0^0$. На основании модели управления программными объектами выбирается начальная программа P^k. Для входных данных из D^k в соответствующих элементах b_p выполняется присваивание $T_p = T_l^k$, $V_p = V_l^k$. Этим и определяется начальное состояние информационной среды B^0.

Используя предыдущий анализ, введем в определение понятие интерфейса в интегрированных комплексах.

Интерфейсом *для программы P^i в ИК называется комплекс методов и средств, обеспечивающий информационное сопряжение текущего состояния информационной среды B и множества данных D^i программы P^i.*

В этом определении отсутствует упоминание о связях между программами, что согласуется с моделью управления программными объектами.

Введенное определение является обобщением интерфейса прикладных программ с БД. В отличие от БД, которая имеет постоянную структуру (связи между отдельными элементами), в информационной среде связи между ее элементами могут динамически меняться, что зависит от конкретно выполняемых программ. Механизмы реализации БД и информационной среды различные. В то же время метод организации интерфейса с прикладными программами аналогичен — через набор специальных подпрограмм.

5.2. ОБЩЕЕ ОПИСАНИЕ МЕТОДОВ И СРЕДСТВ ИНТЕГРАЦИИ

Вопросы интеграции программных средств неявно присутствуют во всех методах, подходах, системах программирования, где происходит регламентация объектов и правил их взаимодействия. В операционных системах прикладным программам предоставляются стандартные механизмы управления вводом-выводом, памятью, задачами и т. д. Благодаря использованию этих механизмов осуществляется совместное функционирование программ в мультизадачной и мультипрограммной средах.

Языки программирования предоставляют множество типов данных, набор операций над данными, механизмы взаимодействия программных объектов. Это является необходимым условием комплексирования объектов.

Различные системы программирования предоставляют разработчику стандартные методы, процедуры, алгоритмы для создания программного обеспечения. Используемые методы, процедуры, алгоритмы обычно содержат правила, необходимые для описания, разработки и объединения различных программных объектов в единый интегрированный продукт.

Несмотря на многообразие рассмотренных аспектов интеграции,

их объединяет то, что большинство методов и средств используется на этапе разработки ПС. Методы построения интегрированных комплексов из готовых программ недостаточно исследованы и инструментальные средства их построения менее разнообразны, чем средства разработки ПС.

Наибольшее развитие интегрированные комплексы получили в связи с широким распространением персональных ЭВМ. К ним относятся интегрированные пакеты для деловых приложений — задачи организационного управления, обработки данных, коммерческих расчетов и т. д. В их состав обычно входят:

текстовый редактор;

средства управления электронными таблицами (ЭТ);

система управления базами данных (СУБД);

пакет деловой графики;

средства коммуникации для функционирования в сети ПЭВМ и др.

В настоящей книге ИК рассматривается как комплекс любых программ. Поэтому интегрированный пакет является одним из его видов. Авторы не ставили перед собой цель — описать как можно больше известных интегрированных средств. Те пакеты или системы, которые будут приводиться, являются иллюстрациями описываемых методов интеграции.

Все методы интеграции условно разделим по двум основным признакам.

1. Различие в этапах разработки ИК. Выделяются методы, применяемые на этапе проектирования программ и на этапе комплексирования готовых программ.

2. Различие в применяемых инструментальных средствах. Выделяется использование готовых интерфейсов и использование средств построения ИК.

Рассмотрим некоторые примеры интегрированных средств.

Методы, применяемые на этапе проектирования программ. Отличительной особенностью этих методов является предопределенность состава программ и совместно используемых типов данных. Главное достоинство — возможность построения интерфейсов, предоставляющих оптимальные механизмы передачи управления и преобразования данных. Недостаток — относительная сложность использования дополнительных программных компонент. Рассмотрим несколько примеров.

Пример 1. Пакет Lotus 1-2-3 [121]. В его состав входят: средства обработки ЭТ; СУБД; пакет деловой графики. Это один из первых интегрированных пакетов.

Пример 2. Система Framework [22]. В ее состав входят: текстовый редактор; средства обработки ЭТ; СУБД; графический пакет.

Для работы с системой используется развитый человеко-машинный интерфейс.

Пример 3. Система Knowledge Man (KMan) [127]. В состав системы входят: текстовый редактор; средства обработки ЭТ; СУБД; пакет деловой графики; средства коммуникации.

Система обеспечивает различные уровни взаимодействия с пользователями в зависимости от их квалификации.

Все эти пакеты характеризуют единые представления данных (в рамках своего пакета) и процедуры управления, связанные с эффективным использованием оперативной памяти и минимизации обращения к дискам. Примером аналогичной системы, разработанной в нашей стране, служит интегрированный пакет Мастер [135]. В его состав входят: текстовый редактор; средства обработки ЭТ; СУБД; средства обработки графиков и рисунков; средства связи с удаленными абонентами.

К интегрированным комплексам следует также отнести Турбо-Системы программирования (Турбо-Паскаль, Турбо-Бейсик, Турбо-Пролог, Турбо-Си). В состав этих систем обычно входят: текстовый редактор; средства обработки файловой системы; транслятор и редактор связей; отладчик.

Использование готовых интерфейсов. Некоторые стандартные пакеты имеют готовые интерфейсы для связи с другими системами. Рассмотрим некоторые примеры.

Пример 1. Пакет Lotus 1-2-3 имеет специальную программу Translate, позволяющую переводить файлы в собственном формате в форматы других систем и обратно. К этим системам относятся также Visicalcud-Base II.

Пример 2. Система Supercalc [51] имеет специальную программу-утилиту SDI для преобразования файлов, записанных в собственном формате, в файлы других систем. К ним относятся: СУБД dBASE-II и Datastar; система Visicalc; пакет Lotus 1-2-3; прикладные программы, написанные на ЯП Бейсик, Паскаль, Кобол.

Пример 3. Система КМап имеет стандартный интерфейс для подключения Си-программ.

Приведенные пакеты и системы реализованы на ПЭВМ. Однако стандартные интерфейсы реализованы и на других классах ЭВМ. Так, на ЕС ЭВМ имеются интерфейсы между СУБД Дисод и ППП Кама, между ППП Кама и СУБД Ока и т. д.

Использование инструментальных средств интеграции. Под инструментальными средствами понимаются средства программирования интегрированных комплексов или средства, обеспечивающие функционирование в интегрированных средах.

Некоторые интегрированные системы имеют средства программирования для разработки прикладных программ, использующих возможности самих систем. К ним, в частности, относятся языки пакетов КМап и Framework.

Пакеты типа ППП Кама предоставляют набор средств для разработки прикладных программ на ЯП высокого уровня. Эти средства включают возможность единого описания данных и стандартные механизмы связи между программами.

Методы комплексирования готовых программ. Как было отмечено ранее, данные методы недостаточно исследованы. Многообразие типов и структур данных, типов внешних устройств, механизмов управления и связи между программами не позволяет создать универсальную

систему интеграции. Существующие методы и средства охватывают только отдельные вопросы интеграции.

Использование готовых интерфейсов. Для комплексирования готовых программ применяются стандартные интерфейсы, описанные ранее. Кроме того, используются интерфейсы, разработанные самим пользователем.

Использование инструментальных средств интеграции. Рассмотрим отдельные примеры.

Возможности управления готовыми программами на уровне функций монитора представляют различные командные языки операционных систем. К ним, в частности, относятся: язык интерпретатора Shall ОС UNIX [12]; языки командных файлов ОС РВ ЭВМ [120]; MS DOS [152]; язык процедур подсистемы диалоговой отладки СВМ ЕС [130] и т. д. На этих языках можно запрограммировать некоторые условия для вызова программ, организации меню, обработки управляющей информации (кодов завершения, состояния устройств и т. д.), вызов самих программ.

На персональных ЭВМ реализованы специальные средства объединения прикладных программ и пакетов — интеграторы. К ним, в частности, относятся WINDOWS, CONCURRENT DOS, GEM, TOPVIEW [22]. Обычно интеграторы работают на уровне ОС — заменяют ее или являются надстройкой над ней. Поэтому прикладные программы должны использовать только стандартные средства ОС ввода-вывода, управления памятью и т. д. Использование интеграторов позволяет:

организовывать многооконный интерфейс с пользователем;

закреплять за отдельными окнами конкретные прикладные программы и выполнять псевдопараллельную обработку;

обмен информацией между прикладными программами через окна;

сделать прикладные программы независимыми от типов используемых на ПЭВМ устройств ввода-вывода — адаптеров клавиатур и терминалов, принтеров.

Выполняя большое число функций по организации управления прикладными программами, интеграторы сравнительно мало обеспечивают возможности информационного обмена в объеме задач, рассматриваемых в настоящей книге. Поэтому для создания универсальных инструментальных средств интегрирования программ интеграторы должны быть дополнены универсальными средствами информационного обмена и средствами автоматизированного выбора программ в ИК для выполнения. Эти средства условно можно определить как средства логического проектирования ИК.

5.3. ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ИНТЕГРИРОВАННЫХ КОМПЛЕКСОВ

В предыдущем изложении методы и средства построения ИК рассматривались в виде отдельных элементов, затрагивающих отдельные частные проблемы. Реальное создание ИК требует применения этих методов и средств в комплексе с учетом их взаимосвязей. Ниже

приводится общее описание методов и средств логического проектирования ИК, разрабатываемых в Институте кибернетики им. В. М. Глушкова АН УССР.

В логическом проектировании ИК выделяются следующие этапы:

1. Формулировка задач, решаемых создаваемым ИК, и выбор существующих компонент.
2. Разработка программных средств для отсутствующих в ИК компонент.
3. Описание базы данных/знаний ИК.
4. Разработка модели информационного сопряжения.
5. Разработка модели управления программными объектами.
6. Реализация среды функционирования ИК.

Рассмотрим эти этапы более подробно.

5.3.1. АНАЛИЗ И ВЫБОР ПРОГРАММНЫХ КОМПОНЕНТ ДЛЯ СОЗДАНИЯ ИК

Задачи, решаемые на данном этапе, относятся к проблеме повторного использования программных объектов. Однако задачи эти сложнее, чем при выборе отдельных модулей. Сложность обусловлена тем, что выбор компонент ИК должен происходить комплексно. В отличие от модулей, которые можно модифицировать в процессе разработки, программы изменять нельзя, и особенности одной из них могут сказываться на функционировании всего ИК. Сложность этой проблемы показывает, что должны существовать специальные инструментальные средства, помогающие выбрать необходимые программные компоненты. Наиболее подходящей для этого является экспертная система (ЭС).

Главное в создании такой ЭС состоит в определении характеристик программных объектов, знаний об этих объектах, на основании которых осуществляется их выбор. Выделим три класса характеристик — функциональные, программно-технические и технологические.

Функциональные характеристики. К ним относятся назначение и возможности данного программного объекта. При создании ЭС важным моментом является иерархическая упорядоченность возможностей по схеме: класс решаемых задач — задача — функции для решения данной задачи — операции, поддерживающие выполнение каждой функции. Наиболее типичный пример для иллюстрации иерархической упорядоченности возможностей представляет СУБД (рис. 5.2). Степень подробности, с которой будут описаны аналогичные схемы программных объектов, влияет на их объективный выбор.

Программно-технические характеристики. К ним относятся характеристики среды функционирования и особенности взаимодействия с другими компонентами. Среда функционирования включает наличие необходимых технических средств, тип операционной системы, особенности выполнения. Взаимодействие с другими компонентами характеризуется наличием стандартных интерфейсов, общим использованием данных, а также связями с системами программирования.

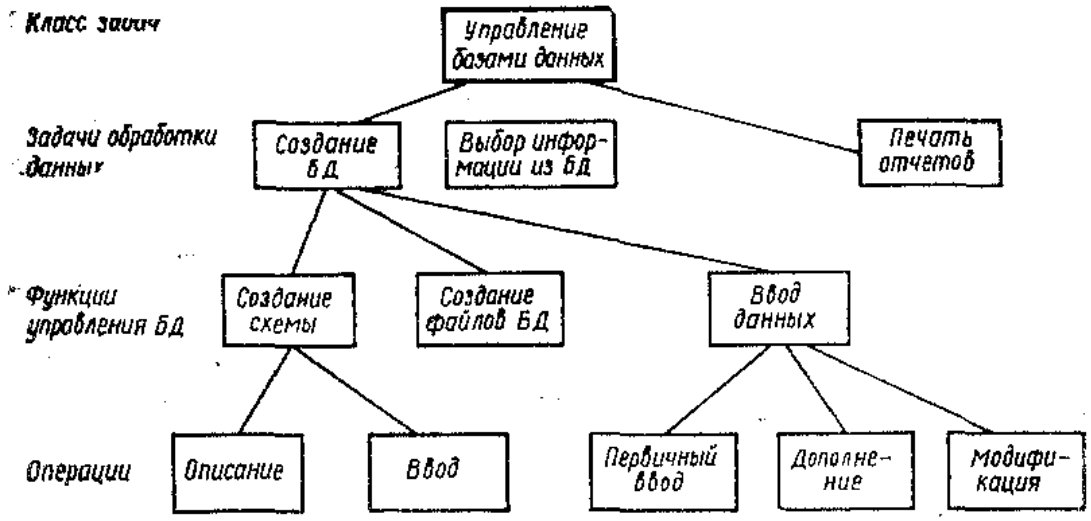


Рис. 5.2. Иерархическая схема возможностей СУБД

Технологические характеристики. Данный класс характеристик неформализуем, что в значительной степени связано с наличием человеческого фактора. К ним, в частности, относятся:

уровень взаимодействия с системой (программист, конечный пользователь и т. д.);

режимы взаимодействия (пакетный, диалоговый, удаленный и т. д.);

наличие или отсутствие собственных средств программирования;

наличие дополнительных сервисных средств;

квалификация пользователя для работы с системой (оператор, системный или прикладной программист и т. д.);

наличие и состав документации;

наличие обучающих курсов;

время освоения данных средств с учетом категории и квалификации пользователя;

временные и надежностные характеристики, показатели качества.

Исходя из разнообразия и большого количества характеристик, можно сделать вывод, что экспертная система не может быть сделана окончательно. В этом ее отличие от обычных автоматизированных информационных систем (АИС), используемых в фондах алгоритмов и программ (ФАП). Эти АИС могут использоваться в качестве отправной точки для создания ЭС, если они содержат необходимую информацию. Развитие экспертной системы связано с определением связей между различными ПС, описанием различных альтернативных путей поиска, формированием численных значений и весовых коэффициентов для отдельных характеристик, построением оценочных функций выбора.

Важным этапом в построении экспертной системы является выбор модели представления знаний. Наилучшим вариантом представления знаний о характеристиках ПС является **фреймовое**.

Информация, о которой говорилось выше, может быть описана как сеть фреймов. Сформулировав запрос на выбор компоненты (например, по функциональным признакам) и постепенно уточняя дополнительные характеристики, мы продвигаемся по фреймовой сети.

С помощью механизмов наследования атрибутов и сопоставления выбирается следующий шаг поиска. Поскольку информация о всех программных объектах входит в единую сеть, тем самым учитываются не только характеристики отдельных объектов, но и совместимость между ними, их связи и т. д. Например, механизм наследования атрибутов позволит контролировать выполнение всех компонентов в единой операционной системе и на однотипных ЭВМ. Более подробно фреймовые модели рассматриваются в [115].

Несмотря на очевидную привлекательность фреймовой модели, она не может применяться на начальном этапе создания ЭС. Это связано со значительной неопределенностью в выборе характеристик программных объектов, их значений, связей и т. д. Поэтому наиболее оптимальным вариантом может служить **продукционная** модель, т. е. представление знаний о ПС в виде правил типа «ЕСЛИ — ТО». Приведем примеры нескольких возможных правил для выбора систем управления базами данных:

ЕСЛИ (требуется СУБД) И (тип ЕС ЭВМ) ТО (СУБД для ЕС ЭВМ)
ЕСЛИ (требуется СУБД) И (тип СМ ЭВМ) ТО (СУБД для ЕС ЭВМ)
ЕСЛИ (требуется СУБД) И (тип IBM PC) ТО (СУБД для IBM PC)
ЕСЛИ (СУБД для ЕС ЭВМ) И (ОС = ОС ЕС) ТО (СУБД для ОС ЕС)
ЕСЛИ (СУБД для ЕС ЭВМ) И (ОС = СВМ ЕС) ТО (СУБД для СВМ ЕС)
ЕСЛИ (СУБД для СМ ЭВМ) И (ОС = ОС РВ) ТО (СУБД для ОС РВ)
ЕСЛИ (СУБД для СМ ЭВМ) И (ОС = UNIX) ТО (СУБД для UNIX)
ЕСЛИ (СУБД для IBM PC) И (ОС = MS DOS) ТО (СУБД для MS DOS)
ЕСЛИ (СУБД для ОС ЕС) И (иерархическая модель) ТО (СУБД ОКА)
ЕСЛИ (СУБД для UNIX) И (реляционная модель) ТО (СУБД INGRES)
ЕСЛИ (СУБД для IBM PC) И (реляционная модель) ТО (СУБД DBASE III)
ЕСЛИ (СУБД для IBM PC) И (требуется интегрированная среда) ТО (пакет Framework).

Эти правила очень простые. Правила для реальной ЭС будут значительно сложнее. Они будут содержать больше характеристик в условной части, потребуются механизмы управления для применения правил, необходимо будет учитывать неопределенность в представлении знаний и т. д. Продукционная модель позволяет добавлять и изменять значения, добавляя или изменяя соответствующие правила.

Форма записи продукционных правил зависит от применяемых инструментальных средств построения экспертных систем — оболочек ЭС, языков представления знаний, языков программирования. Рассматриваемая ЭС имеет особенность, связанную с тем, что формировать базу знаний может не специалист по инженерии знаний, а конечный пользователь. Наиболее вероятно, что это будут специалисты из

фондов алгоритмов и программ. Поэтому им нужны доступные и простые в использовании инструментальные средства. Для этих целей подходят пакеты создания прикладных экспертных систем. В качестве примера рассмотрим пакет Rule Master [135]. Он обеспечивает несколько уровней представления знаний. Один из них — уровень конечного пользователя. Знания представляются в табличной форме и для рассмотренных выше правил связи с СУБД будут иметь вид, приведенный в табл. 5.1.

Таблица 15.

Тип ЭВМ	ОС	Характеристика I	. . .	Характеристика N	Дальнейшие действия (выбор СУБД)
ЕС ЭВМ	ОС ЕС	. . .		. . .	ОКА
ЕС ЭВМ	СВМ ЕС	. . .		. . .	. . .
СМ ЭВМ	ОС РВ	. . .		. . .	. . .
СМ ЭВМ	UNIX	. . .		. . .	INGRES
IBMPC	MSDOS	. . .		. . .	dBASE-III

С помощью специальных средств происходит описание этой таблицы, а также действий для ввода данных, соответствующих отдельным столбцам. В качестве дальнейших действий может быть переход к анализу таблиц, содержащих дополнительные данные, необходимые для уточнения поиска.

После описания и ввода табличного представления средства Rule Master проверяют знания на полноту и непротиворечивость и строят дерево вывода. При работе ЭС осуществляет ввод данных (с терминала пользователя, из базы данных), вычисляемых прикладной программой, и ищет подходящую строку в таблице. Если строка найдена, то выполняются дальнейшие действия (вывод результата или переход к анализу следующей таблицы). Необходимо отметить, что при таком подходе изменение и дополнение знаний связано с изменением или дополнением строк и столбцов таблицы.

5.3.2. РАЗРАБОТКА ПРОГРАММНЫХ КОМПОНЕНТ ИК

Необходимость в этом этапе возникает при отсутствии всех готовых компонент или если они не удовлетворяют требуемым условиям. В основу разработки отдельных программ ИК положен модульный принцип. Суть модульного принципа рассмотрена в предыдущих главах. Вопросы создания межмодульных интерфейсов проанализированы в гл. 3, методы построения и управления модульными структурами программ — в гл. 4. Технологические аспекты применения модульного принципа будут приведены в гл. 7.

Остановимся на особенностях данного метода по сравнению с разработкой автономных программ.

1. При проектировании необходимо учитывать взаимосвязи не только внутренних объектов (модулей, сегментов), но и связи с другими компонентами ИК.

2. Поскольку остальные компоненты интегрированного комплекса известны, то структуры внешних данных разрабатываемой программы могут быть максимально согласованы со структурами данных других программ.

3. Средства реализации интерфейсов могут входить в состав модулей программы.

4. Функции монитора программы, если он необходим, будут упорядочены в связи с существованием монитора ИК.

5. Если необходимо создание нескольких программ, то может возрасти сложность технологических аспектов управления разработкой ИК.

5.3.3. ОПИСАНИЕ БАЗЫ ДАННЫХ ИК

Данные, входящие в базу данных (БД) ИК и описываемые выражением (5.1), включают множества данных отдельных программных компонент D^i и множество управляющих данных D^c.

Множества D^i, в свою очередь, состоят из внешних и внутренних данных P^i, влияющих на функционирование других компонент (например, данные сред ЯП, на которых написаны модули программы P^i).

Для описания БД ИК необходимо описать все переменные, входящие в ее состав. Описание переменной $d^i_j \in D$ состоит в описании ее имени N^i_j, типа данных T^i_j и в определении соответствия между рассматриваемой переменной и областью памяти, где располагается ее значение.

Для описания переменных используется язык описания типов данных (ЯОТ). В дальнейшем мы не будем останавливаться на синтаксических конструкциях языка, а рассмотрим только его семантику. ЯОТ должен удовлетворять следующим требованиям:

допускать описание всех типов данных на единой методологической основе независимо от ЯП, на которых разрабатываются модули программ из ИК;

отражать современные тенденции в теории типов данных;

описывать переменные и их типы без привлечения дополнительных языковых средств;

содержать информацию для размещения данных в БД;

обладать простотой в изучении и гибкостью в использовании.

Анализируя эти требования, можно отметить, что ЯОТ близок к средствам описания данных в современных ЯП высокого уровня — Паскаль, Модула-2, Ада. Методологической основой ЯОТ служит теория структурной организации данных в ее наиболее полном виде. Средства конструирования типов аналогичны соответствующим средствам указанных ЯП. Однако ЯОТ имеет ряд особенностей, связанных с использованием его в проектировании интегрированных комплексов. В частности, отметим следующие.

1. Для описания управляющих переменных (множество D^c) введены специальные типы данных. Например, к ним относится тип

PROGVAR, рассмотренный в гл. 2 при описании модели управления программными объектами.

2. Расширены средства описания файлов. Файлы могут содержать фиксированное и переменное число записей. Записи могут быть постоянной или переменной длины, иметь регулярную или произвольную структуру. В файл могут явно входить разделители записей и отдельных полей записи. Средства языка должны обеспечивать описание указанных характеристик.

3. Введен специальный тип данных для описания командных строк. Командная строка может содержать неопределенное число параметров (возможно, пустое) с произвольным порядком их следования. Эта особенность должна быть отражена в описании рассматриваемого типа.

4. Введена возможность указания начальных значений переменных. Входные переменные ИК, отличные от файлов, должны быть инициализированы. Это достигается путем присваивания переменным начальных значений.

Язык описания типов данных позволяет описать следующую информацию.

Имя описания. В качестве имени описания принимается имя соответствующей программы.

Раздел констант. В этом разделе константам присваиваются символические имена. Данные константы используются при обработке конструкций языка, в которых вместо непосредственных значений применяются символические имена.

Раздел описания типов данных. В этом разделе описываются типы данных и их иерархия аналогично соответствующим описаниям в ЯП Паскаль, Модула-2. В описание типа может включаться признак разделителя. В этом случае предполагается, что за всеми переменными данного типа находится рассматриваемый разделитель.

Раздел описания переменных. Приводятся списки переменных с указанием их типов. Переменным могут быть присвоены начальные значения. В частности, указываются имена файлов. Для некоторых переменных может присутствовать признак наличия разделителя.

Конец описания. Это — оператор завершения описания данных для программы.

Транслятор с ЯОТ переводит описание в набор внутренних таблиц, которые служат основой для формирования БД, доступа и обработки данных в рамках реализации моделей информационного сопряжения и управления программными объектами. В процессе совершенствования методов построения ИК структура ЯОТ и средства его обработки, возможно, будут изменяться.

5.3.4. РАЗРАБОТКА МОДЕЛИ ИНФОРМАЦИОННОГО СОПРЯЖЕНИЯ

Модель информационного сопряжения разрабатывается исходя из понятия интерфейса, введенного в п. 5.1. Фактически там же описан алгоритм для создания интерфейсов, суть которого состоит в следующем.

1. Все данные, входящие в БД ИК, делятся на классы эквивалентности. Каждый класс определяет совокупность данных, принадлежащих разным множествам D^i и имеющим одинаковое семантическое содержание.

2. Перед вызовом очередной программы происходит преобразование данных, входящих в текущее состояние информационной среды, в представление согласно описанию данных для этой программы. При этом используется набор стандартных функций преобразования типов данных. Типы данных описываются средствами ЯОТ. Внутреннее представление описания служит схемой доступа к данным для процедур, реализующих функции преобразования типов данных.

3. Действия, описанные во втором пункте, повторяются для каждой вызываемой программы.

Рассматриваемый алгоритм основывается на описаниях данных и правильном разбиении их на классы эквивалентности. Средства описания данных приведены выше.

Рассмотрим средства выделения классов эквивалентности. Для этой цели используется язык описания классов эквивалентности (ЯОК). Он оперирует только именами переменных и их семантическим содержанием. Как и раньше, синтаксические конструкции мы не рассматриваем. Описание на ЯОК состоит из следующих разделов.

Имя описания. В качестве имени описания используется символическое наименование проектируемого интегрированного комплекса.

Раздел описания программ. В этом разделе указывается список имен программ, входящих в ИК. Если необходимо, для некоторых программ приводится список переменных, определяющих командную строку.

Раздел описания переменных. Состоит из нескольких подразделов согласно числу программ, входящих в ИК. Каждому подразделу присваивается имя соответствующей программы. В него входит список имен переменных с указанием их видов и семантического содержания. Переменные делятся на два вида— входные и выходные. Семантическое содержание представляет собой предложение на обычном языке, определяющее назначение данной переменной для выбранной программы.

Раздел описания классов эквивалентности. Состоит из нескольких подразделов согласно числу построенных классов, каждому присваивается имя соответствующего класса эквивалентности. Подраздел состоит из списка пар — имя переменной, имя программы, в которой она используется. Данный раздел составляется и описывается разработчиком ИК.

Раздел описания ключевых слов. Этот раздел является альтернативой разделу описания классов эквивалентности. Он содержит список слов, которые объявляются ключевыми. По этим словам осуществляется автоматический поиск в разделе описания переменных. Анализируется семантическое содержание переменных и строятся из них списки, связанные с каждым ключевым словом. Затем в процессе диалога происходит корректировка полученных списков и выделение классов эквивалентности. Данный раздел целесообразно включать при большом числе переменных, когда ручная обработка затруднительна.

Необходимо отметить, что под ключевым словом понимается строка символов, которая в общем может содержать несколько слов.

Конец описания. Для этой цели используется оператор конца описания.

Транслятор с ЯОК формирует набор таблиц, связанных с таблицами, являющимися результатом работы транслятора с ЯОТ. Благодаря этому формируется внутренняя модель представления данных БД ИК. Согласно этой модели функционирует интерфейс, обеспечивающий информационное сопряжение отдельных компонент интегрированного комплекса.

5.3.5. РАЗРАБОТКА МОДЕЛИ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ОБЪЕКТАМИ

В гл. 2 подробно рассмотрено назначение, содержание и использование модели управления программными объектами. Здесь мы приведем общее описание языка для обработки данной модели. Как и прежде, синтаксические конструкции языка не рассматриваются.

Язык описания модели управления программными объектами (ЯОУ) имеет следующую структуру.

Имя описания модели. В качестве имени описания используется символическое имя, присваиваемое создаваемому ИК. Описание состоит из нескольких разделов описания условий выполнения программы согласно количеству компонент, входящих в ИК.

Раздел описания условий выполнения программы. В качестве имени раздела используется имя программы. В раздел включаются описания действий, которые необходимо выполнить до вызова программы и после ее завершения. Выполняемые действия делятся на проверку условий и изменение базы данных. Проверка условий выполняется с помощью набора стандартных предикатов. В качестве примера приведем предикаты проверки:

готовности (наличия) данных;

отсутствия данных;

выполнения определенных программ, входящих в ИК;

вычисления всевозможных логических выражений, заданных над элементами из базы данных ИК;

правильности функционирования (наличие зацикливания, аварийного завершения предыдущей программы и т. д.);

завершения работы (описывается для каждой задачи, решаемой в рамках создаваемого ИК).

В качестве функций изменения БД могут быть:

установка или сброс признака готовности данных;

изменение значений некоторых элементов (например, присваивание «программной» переменной имени очередной выполняемой компоненты);

перемещение данных (копирование файлов и т. д.);

формирование признака завершения работы.

Программа готова для выполнения, если все предшествующие предикаты истинны и все предшествующие функции успешно выполнены.

Конец описания. Завершает описание специальный оператор окончания.

Транслятор с ЯОУ обрабатывает описание модели и переводит его в систему правил-продукций (пример таких правил приведен в гл. 2). Затем в зависимости от режима работы система продукций оформляется в виде текста программы на ЯП Пролог или кодируется в виде допустимых правил одного из пакетов для создания экспертных систем.

Особенности предикатов и функций ЯОУ состоят в том, что они связаны с данными, входящими в БД ИК. Поэтому условие типа $A < B$ будет записываться в виде $f\,\mathrm{get}\,(A) < f\,\mathrm{get}\,(B)$, где функция $f\mathrm{get}(x)$ выбирает значение переменной с именем x из БД ИК. В ЯОУ входит небольшое количество базовых предикатов и функций, а остальные могут быть запрограммированы разработчиком ИК в виде их комбинации.

Одним из уровней представления модели управления программными объектами может быть непосредственное составление разработчиком ИК программы на одном из ЯП. В этом случае необходимо использовать стандартные функции доступа к данным из БД ИК.

5.3.6. СОЗДАНИЕ СРЕДЫ ФУНКЦИОНИРОВАНИЯ ИК

Предыдущие этапы логического проектирования ИК обеспечивают подготовку отдельных элементов интегрированной среды функционирования.

1. **База данных ИК** формируется на основе описаний, составленных на языке ЯОТ. Реальное расположение объектов базы данных определяется требованием на их обработку. Так, файлы, обрабатываемые программами, не копируются, а во внутренних таблицах отмечаются ссылки на них. Командные строки формируются непосредственно перед вызовом соответствующих программ. Для управляющих переменных выделяется область памяти, формируемая на диске, и т. д. Для доступа к базе данных ИК используется набор специально разработанных процедур, включающий функции занесения, выбора, изменения данных различных типов.

2. **База знаний ИК.** Ее основой является описание модели управления программными объектами. Результатом обработки описания служит текст программы на ЯП Пролог или набор продукционных правил для пакета программ построения экспертных систем. В первом случае происходит формирование загрузочной программы, которая включается в состав компонент ИК. Во втором случае в качестве компоненты ИК принимается экспертная система, построенная на основе набора продукционных правил. Основными функциями данной компоненты являются выбор имени очередной выполняемой программы ИК; изменение в случае необходимости информации в БД ИК; определение условий завершения задач, решаемых в рамках ИК.

Для доступа к данным используется тот же набор процедур, как и при обработке БД ИК. Предикаты и функции реализованы с применением этих процедур.

3. **Интерфейсы.** Основой реализации интерфейсов служит описание классов эквивалентности. Перед вызовом выбранной программы выполняются необходимые преобразования данных текущего состояния информационной среды согласно описанию типов данных соответствующей компоненты. Для выполнения этих операций используется набор стандартных функций преобразования типов данных. Функции реализованы с учетом реального представления информации и ее расположения в файлах или оперативной памяти.

4. **Средства управления ИК.** К ним относится монитор ИК, выполняющий: организацию диалога с пользователем; вызов программных компонент ИК для выполнения; контроль состояния среды функционирования и завершение работы.

Монитор служит надстройкой над компонентами ИК и является программой универсального типа.

Созданный на этапе логического проектирования, ИК функционирует согласно следующей схеме алгоритма:

1. Вызов монитора ИК. Монитор вызывается как обычная программа в рамках используемой ОС.

2. Вводится символическое имя ИК. Согласно этому имени активизируются соответствующие описания и выполняются действия по подготовке БД ИК к использованию.

3. Вызов компоненты определения имени текущей выполняемой программы. Данная компонента (Пролог-программа или экспертная система) выполняет действия согласно модели управления программными объектами и определяет имя программы или формирует признак завершения работы.

4. Анализ признака завершения монитором. Если он установлен, то работа ИК прекращается. В другом случае для выбранной программы выполняются необходимые преобразования данных.

5. Из монитора формируется вызов выбранной программы. После окончания программы повторяются действия третьего этапа. Необходимо отметить, что на том же этапе выполняются операции, определенные в модели управления программными объектами и описанные после вызова программы.

Рассмотренный метод построения ИК ориентирован на вопросы логического проектирования комплекса. Поэтому описанные средства могут использоваться совместно с одной из систем-интеграторов, описанных ранее. В этом случае монитор ИК упростится и часть его функций будет возложена на выбранную систему. Алгоритм функционирования комплекса будет зависеть от применения конкретного интегратора.

Глава 6

ВОПРОСЫ ТЕХНОЛОГИИ СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

6.1. ЭЛЕМЕНТЫ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

В последние годы в области программирования интенсивно проводились работы в плане совершенствования программного обеспечения (ПО) больших и малых машин, создания конкретных «уникальных» систем ПО реализации отдельных задач областей народного хозяйства, создания систем поддержки разработки ПО и технологических комплексов для обеспечения процессов разработки ПО общего назначения. Однако по-прежнему остаются нерешенными важные проблемы программирования:

постоянный рост стоимости ПО (так, в США к 1980 г. стоимость ПО составляла 3 млрд дол., а к 1990 г.— 30 млрд дол.) [108];

невысокая производительность процесса создания ПО (в США ежегодный прирост продуктивности ПО составляет 3—8 %, а ЭВМ — 40 %) [174];

низкое качество ПО, несмотря на возросшие к нему в последнее время требования (так, ВВС США предъявляют жесткие требования к ПО — отсутствие сбоев и корректировок) [108].

Эти данные говорят о продолжающемся кризисном состоянии в области ПО, суть которого заключается в значительном отставании программирования как технологического процесса от технологии разработки и производства ЭВМ.

Увеличение годового роста спроса на ПО до 24 %, что соответственно влечет за собой увеличение на 12% персонала (в США оно фактически растет на 3—4 % [174]), еще более обостряет эти проблемы.

Указанное отставание объясняется прежде всего тем, что, согласно [108], в программировании отсутствует единое понимание сущности технологических процессов разработки ПО и средств их автоматизации (как это сделано в машиностроении — ГОСТы, ЕСТПП), а также низким инженерным уровнем проектирования и разработки, который вызван плохим качеством планирования и контроля процесса разработки ПО, отсутствием критериев и средств оценки количественных и качественных характеристик ПО и недостаточным проведением работ по стандартизации и унификации в области программирования, особенно в части повторно используемых заготовок (программ, модулей, процессов и т. п.), подобно тому как это используется во всех отраслях промышленности.

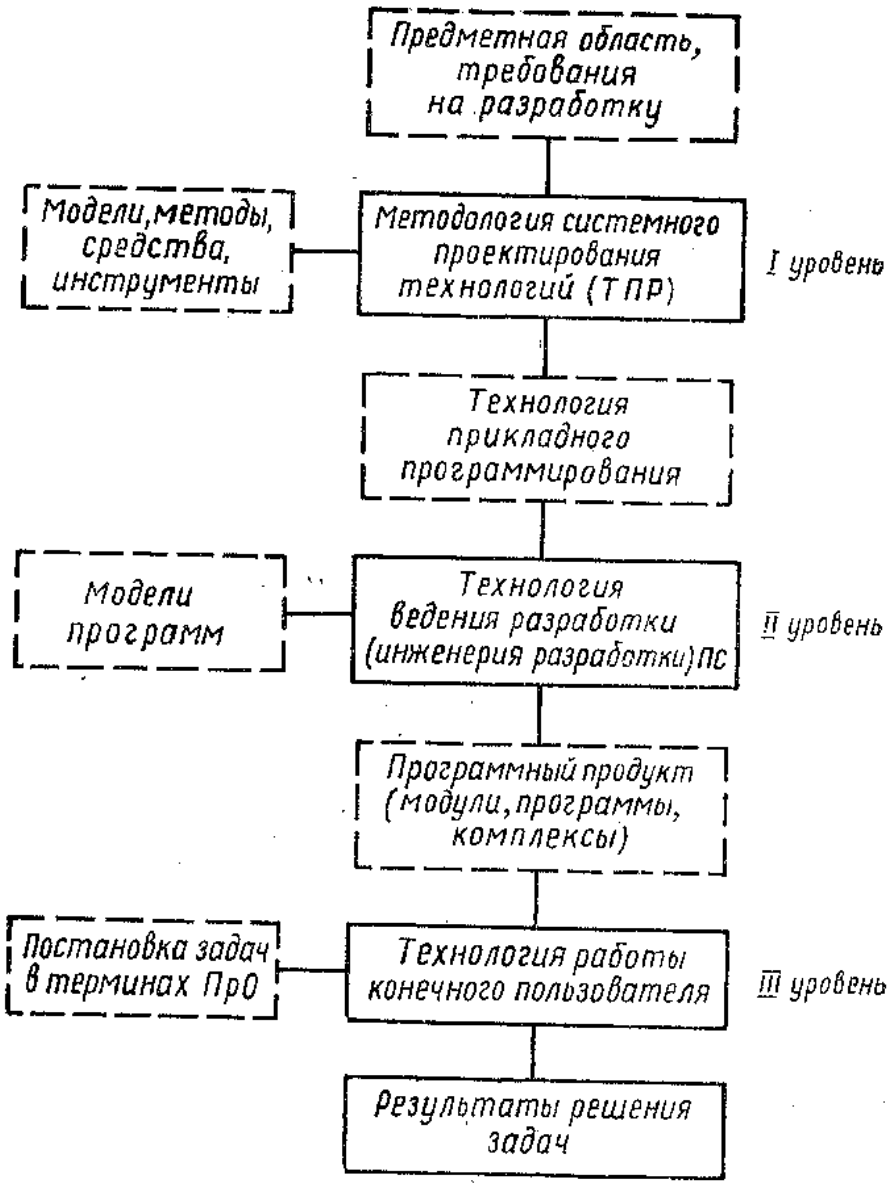


Рис. 6.1. Уровни технологии программирования

В связи со сказанным можно считать, что первоочередной и наиболее актуальной задачей современного программирования является не столько создание масштабных «уникальных» программных систем реализации конкретных функций предметных областей, а сколько создание эффективных технологий и инструментов, поддерживающих процессы инженерного проектирования и разработки их программного обеспечения.

Под технологией программирования понимается совокупность принципов, методов, подходов, упорядочивающих процесс разработки программного обеспечения с заданными характеристиками на всех этапах создания ПО. В такой постановке эффективность процесса создания программного продукта зависит от обоснованности применения методов проектирования автоматизируемой предметной области, принятой организации и методов инженерного управления разработкой ПО (планирование, нормирование, учет и др.), а также

от степени оснащенности процессов разработки средствами автоматизации и готовыми элементами повторного использования.

Таким образом, разработке ПО должно предшествовать первоначальное определение или выбор готовой технологии, учитывающей специфику и функции реализуемой предметной области (ПрО). Именно такой подход традиционно существует в промышленности при разработке технических изделий, о чем свидетельствует большой прогресс в области роста продуктивности современных ЭВМ (на 40 %). Для лучшего понимания сущности технологии программирования необходимо привести определения ее основных элементов и их назначение.

В технологии программирования четко выделяются три основных уровня технологий (рис. 6.1); системного проектирования; прикладного программирования; конечного пользователя.

Первый уровень — это технология подготовки разработки (ТПР), направленная на определение модели предметной области и реализуемого программного объекта. Для объекта вычленяется набор его состояний жизненного цикла, методов, средств и инструментов преобразования этих состояний с целью получения его в виде программного продукта. Эту технологию определяют высококвалифицированные специалисты, знающие предметную область, создаваемый объект разработки, а также пути его разработки. Результатом их деятельности является продукт — конкретная прикладная технология, по которой ведется разработка множества программ из заданного класса. На этом уровне технологии определена последовательность действий и инструментов, которые надо произвести для получения соответствующего прикладного программного продукта.

Второй уровень технологии — подходы к разработке прикладных программных элементов по заданной на уровне ТПР технологии прикладного программирования. Этот уровень предусматривает применение методов планирования (составления план-графиков), контроля и регулирования хода разработки и оценки результатов труда для достижения высокого качества продукта. Разработку проводят прикладные программисты. Результатом их деятельности являются программный продукт, готовый к выполнению, и технология решения задач конкретным пользователем.

Третий уровень — технология конечного пользователя, в соответствии с которой он осуществляет постановки задач и получение результатов решения. Основное требование, предъявляемое к этому уровню технологии, состоит в том, чтобы пользователь работал с прикладной системой в терминологии предметной области.

Главными элементами программной технологии являются: объект разработки; программные методы и средства, объединенные в технологические процессы и линии; инженерные методы управления разработкой.

Объект разработки — это программный продукт, реализующий определенные функции (задачи) предметной области, процесс проектирования и разработки которого осуществляется соответствующими методами и программными средствами.

Объектом разработки может быть: модуль, программа, комплекс программ, пакет прикладных программ, система и др., т. е. объект либо сам является отдельной конструктивной единицей разработки, реализующей элементарную функцию ПрО, либо состоит из взаимосвязанного их набора.

Как показывает анализ современной литературы [28, 66, 125, 142, 144 и др.], для объекта разработки первоначально формируется его модель, в которой специфицируются реализуемые функции и элементы данных, детализируются их связи и отношения, т. е. для объекта определяются его спецификация или прототип.

К объекту разработки в техническом задании предъявляются определенные требования к составу реализуемых функций, ограничениям на такие характеристики, как работоспособность,что означает отсутствие ошибок при выполнении функций (свойство надежности), быстродействие, память, удобство общения (свойство эргономичности), мобильность (свойство переносимости из одной среды в другую) и др.

В зависимости от требований к условиям функционирования объекта (работать с большим быстродействием в условиях реального времени или с большой точностью в условиях космических, военных систем и т. п.) требования к свойствам разных программных объектов отличаются и влияют на организацию процесса разработки [1, 33, 143 и др.].

Любой объект имеет начальное (исходное), промежуточные и конечное состояния [17, 104, 108 и др.]. Начальное состояние — это исходная модель объекта. Промежуточное — измененное состояние, отличное от начального и конечного состояний объекта, полученное на определенном этапе жизненного цикла под воздействием соответствующих данному этапу программных методов и средств. Промежуточным состоянием объекта по крупному в соответствии с ГОСТами ЕСПД являются: эскизный, технический, рабочий проекты.

Конечное состояние объекта — программный продукт, готовый для выполнения требуемых от него функций. Конечным состоянием считается опытный образец, передающийся либо в опытную эксплуатацию, либо на тиражирование (производство).

Следует отметить, что еще не стабилизировался формальный аппарат задания начального состояния объекта, а именно его модели, чего нельзя сказать о возможностях описания промежуточных состояний, для задания которых используются языки проектирования и программирования различного уровня, в том числе языки спецификаций.

Метод разработки (программный метод) — это способ и средства достижения той цели, которая ставится перед объектом разработки. Метод определяет стратегию проектирования или разработки.

Наиболее распространенными методами проектирования и разработки являются: сверху-вниз, снизу-вверх, модульный, метод расширения ядра, метод сборочного программирования.

При нисходящем (сверху-вниз) проектировании объекта после определения требований к объекту формируется его функциональная

и системная архитектуры. В них декомпозированы все задачи ПрО и построена модель объекта. Задачи, в свою очередь, развиваются до понятий и функций объекта, выражаемых в базовых терминах рассматриваемой ПрО. Каждая функция объекта разрабатывается последовательным уточнением до определения элементов системной архитектуры.

Восходящий метод (снизу-вверх) является обратным нисходящему и начинается с уровня элементарных базовых понятий ПрО и формирования из них более крупных понятий, приводящих в конечном итоге к определению некоторой функции ПрО. Для сформированной функции подбирается или разрабатывается один из программных элементов, модуль, модель программы, макрос, заготовка и т. п. В этом плане данный метод можно считать методом от готового.

Развитием восходящего метода является метод прототипирования [195], при котором для объекта вначале создается «грубый» его прототип из готовых компонентов. При этом от программ требуется, чтобы они соответствовали функциям объекта без учета эксплуатационных характеристик, т. е. цель прототипирования состоит в том, чтобы отработать «каркас» объекта и его функциональные возможности, а затем улучшить характеристики и свойства компонентов. Данный метод в настоящее время начинает широко применяться в практике программирования, так как он дает возможность быстро опробовать и отработать требования заказчика к программному объекту вместе с разработчиками.

Метод расширения ядра характеризуется начальным выделением множества вспомогательных функций. Наиболее эффективным методом выделения является анализ используемых данных и определение модулей, обрабатывающих различные информационные структуры. Для этого может быть использован метод Джексона, в основе которого лежит принцип соответствия организации программы согласно этапам преобразования обрабатываемых данных.

Несмотря на различие этих методов в стратегиях проектирования, общее, что их объединяет,— это модульное или блочное (программное [73]) представление объекта разработки. Суть метода модульного программирования состоит в декомпозиции исходной задачи ПрО на отдельные функции вплоть до элементарных, каждой из которых в общем случае сопоставляется программа или модуль. Каждый модуль должен обладать интерфейсом для его связи с другими модулями и компонентами. Применение данного метода по сравнению с другими дает значительные преимущества в плане организации и управления разработкой, но вместе с тем требует создания определенных механизмов языкового и программного характера для решения проблем интерфейса при сборке модулей и программ в более сложные программные структуры. Расширением данного метода является метод сборочного программирования.

Таким образом, рассмотренные наборы методов проектирования и программирования взаимосвязаны и используют друг друга. Так, в случае применения восходящего метода проектирования для систем

обработки данных на разных этапах жизненного цикла программного объекта использовались методы структурного и модульного программирования, расширения ядра и др.

Технологический процесс — это взаимосвязанная последовательность операций разработки объекта. Процесс предназначен для перевода объекта из одного состояния в другое соответствующими методами и программными средствами.

Характерной особенностью многих технологических комплексов является регламентация этапов жизненного цикла (ПРОМЕТЕЙ, ТКП, РТК и др.). В этих комплексах технологические процессы не связаны со спецификой функций ПрО (например, таких, как функций ввода, вывода данных и т. п.). Они обеспечивают поддержку процессов разработки ПС универсальными средствами автоматизации, ориентированными на любые виды программ. Другая интерпретация процессов заключается в том, что они позволяют реализовать набор типовых функций автоматизируемой ПрО, относящейся к СОД, АСНИ, САПР и др. Такой процесс становится типовым и может использоваться как объект сборки в технологиях, где он является необходимой составной частью. Типовые процессы вместе со специализированными процессами образуют линию разработки программ по технологии функционально-ориентированного типа.

Технологическая линия задает технологию разработки функции объекта, представленную совокупностью автоматизированных процессов, последовательно и систематически преобразуемых состояния объектов, включая заключительное состояние — готовый программный продукт. Особенностью линии является отражение определенных функций ПрО, реализуемых в виде программ с заданными показателями качества.

Для простых объектов количество процессов в ТЛ меньше, чем для сложных. Независимо от сложности объекта основным условием выполнения процессов является их автоматизация. Процессы имеют модельное представление, базирующееся на модели жизненного цикла объекта разработки.

Динамика изменения может быть представлена в ТЛ последовательностью процессов, переводящих объект из одного состояния в другое путем использования средств автоматизации процесса. В этом плане каждая конкретно разработанная ТЛ определяет множество допустимых состояний объекта, а на средства автоматизации возлагается функция слежения за переходом в недопустимые состояния.

Инструмент — это программное, языковое или методическое средство, применяемое для задания состояния объекта в некотором законченном виде. В зависимости от этапа жизненного цикла и состояния объекта инструментами для работы с ним могут быть языки, трансляторы, генераторы и т. п.

Для описания модели пограммного объекта используются имеющиеся формальные или полуформальные средства (графические, языковые) спецификаций требований [143].

К графическим средствам относятся: таблицы для описания конечных функций; решающие таблицы и структуры (матрицы, векторы

и т. п.); графовые структуры (диаграммы взаимодействий и состояний Петри и др.); схемы HIPO, SADT и др.

К языковым — языки спецификаций [1, 4, 33, 123, 143 и др.] общего и специального назначения, ориентированные на специальные ПрО (языки описания моделей АСУ в системах АРИУС, ISDOS). К языкам спецификаций общего назначения относятся GDL, PDL, PSL, PSA и др. Они позволяют представлять «каркас» или схему объекта посредством структур управления (ветвления, цикла, вызовов процедур).

Процесс преобразования модели объекта из одного состояния в другое не является полностью автоматизированным, и имеется ряд систем, специально предназначенных для реализации конкретных предметных областей.

Наибольший набор инструментов создан для этапа программирования. К ним относятся языки программирования и соответствующие им системы программирования. Более поздние языки программирования, такие, как АДА, Паскаль, Си, содержат средства описания абстрактных структур данных и задания структур сложных программ из модулей.

Управление разработкой. Каждая инженерная дисциплина базируется на принципах и методах конструирования (разработки) и промышленного производства продуктов, которые затрагивают как организационные, так и технические аспекты производства. В этом случае программную технологию в последнее время рассматривают с позиций инженерной дисциплины [2, 17, 18, 32, 33, 73, 82, 108, 109, 188 и др.]. Исходя из анализа этих работ заметим, что основными вопросами управления разработкой программных объектов как инженерии ведения разработки являются:

организация коллектива разработчиков (состав, структура, квалификация и др.);

планирование работ и трудозатрат и обеспечение роста производительности труда;

контроль хода разработки и оценка проектных решений в ходе разработки программных продуктов;

экономические вопросы (стоимость, ценообразование, стимулирование и др.).

6.2. ОСНОВНЫЕ ПУТИ РАЗРАБОТКИ ПРОГРАММНЫХ ТЕХНОЛОГИЙ

Выше рассмотрены основные понятия и элементы технологии программирования. Многообразие методов, подходов, инструментальных средств и т. д. порождает многообразие программных технологий. Однако есть общее, что объединяет их. **Во-первых,** общие этапы жизненного цикла разработки ПО. **Во-вторых,** общие пути разработки программных технологий. Первая особенность рассмотрена в гл. 1, а вторую рассмотрим здесь.

Разработка программной технологии традиционно осуществлялась двумя путями. На первом фиксируется изначально набор инструмен-

тальных средств, на базе которого строятся процессы разработки программных систем. Во втором случае, исходя из общих и специфических свойств технологических процессов реализации функций ПрО, разрабатывается или выбирается набор инструментов, необходимых для эффективной их реализации.

Первый путь является наиболее типичным. Вся история развития программирования характеризовалась именно тем, что пользователи (разработчики ПС) пытались применять вновь появляющиеся средства автоматизации (языки программирования, системы программирования и управления базами данных, пакеты прикладных программ и др.) под программные нужды, формируя при этом свои стили и технологии их применения.

При этом зачастую оказывалось, что данные средства не полностью удовлетворяли требованиям разработок и разработчики занимались их развитием и совершенствованием в нужном направлении.

К системам, использующим данный метод, относятся и современные хорошо развитые и широко используемые универсальные технологические системы и комплексы (РТК, ПРИЗ, ПРОМЕТЕЙ, ТКП и др.). В них нет четко выраженной направленности на специфику конкретной ПрО, но присутствует общая регламентация процессов разработки ПС и хранения истории ведения разработки. Задача пользователя этих систем состоит в том, чтобы использовать предлагаемую технологию и средства ее инструментальной поддержки при реализации конкретной ПрО. Однако их применение ограничено из-за отсутствия средств для представления важных функций ПрО. Например, средствами указанных технологий не удается эффективно реализовать прикладное программное обеспечение СОД, так как оно требует для своей работы среду СУБД.

Второй путь состоит в том, что технологические процессы создания программного продукта сводятся к описанию структуры и свойств этого продукта, направленного на реализацию функций ПрО, а процесс получения собственно продукта осуществляется автоматически. Данный подход принят за основу автоматизации конкретных предметных областей: АСУ, СОД, АИС и др.

Рассмотрим основные системы автоматизированной поддержки разработки указанных предметных областей.

Система АРИУС [31] реализует архитектурный подход к проектированию АСУ. Автоматизация системы начинается с анализа документооборота, определения функций и проектирования информационного обеспечения в виде информационно-логической модели.

Разработчик АСУ средствами системы АРИУС осуществляет ранжирование переменных массива описания данных; присваивание им отношений типа «синоним» и «часть — целое» между переменной и ее аргументом или между переменными одного ранга; создание словаря переменных и их спецификаций. В результате создается тезаурус, представляющий собой совокупность списков характеристик и признаков элементов системы, связей и отношений между ними. Спецификация описывает функциональные отношения, упорядоченные в соответствии с иерархией переменных в тезаурусе.

Проектируемая средствами АРИУС система управления обеспечивает удобный сервис пользователю, предоставляя ему возможность по вводу и учету условно-постоянной информации, хранению, обработке и выдаче данных в АСУ.

Система ISDOS [65] предназначена для документированного проектирования СОД. В ней реализованы язык постановки задач (PSL) и язык описания свойств объектов и их отношений (PSA).

Основными элементами языка PSL являются:

организационный объект для описания окружающей среды, в которой будет работать система;

информационный объект для описания таких элементов, как вид и связи данных, преобразование данных;

объект управления проектом для описания информации о ходе проектирования.

С помощью языка PSA и анализатора с него осуществляются сбор информации о системе, ввод описаний и постановок задач в PSL, включение информации в базу данных, генерация выходных данных. Данная система повышает уровень разработки систем типа АСУ на всех этапах ее создания, предоставляя различные сведения о ходе разработки.

Система ПЛЮС [149] представляет собой комплекс взаимосвязанных интегрированных компонентов, образующих узкоспециализированную среду для проектирования, программирования и эксплуатации СОД. Связывающим звеном разработки является база данных (словарь данных), связывающая описание разрабатываемой системы СОД. Она описывается в специализированном языке ПЛЮС, содержащем традиционные средства ЯП, а также возможности связи с СУБД, генераторами отчетов и др.

В системе реализована технология работы или технологическая схема работы, обеспечивающая проектирование информационной базы СОД и функциональной части системы.

Результатом первого этапа проектирования является информационная метабаза (база проекта) СОД, включающая в ее наборы данные и информацию об объектах СОД. Каждое данное СОД имеет три уровня представления: на входе, выходе и в информационной базе. Подсистема управления проектом позволяет держать сведения о ходе разработки СОД в актуальном состоянии, создавать фрагменты (и в целом) документации на систему.

На втором этапе осуществляется программирование функций в виде прикладных программ, использующих имена данных, и таблиц словаря справочника. В этом и состоит существенное отличие традиционных методов разработки СОД от методов системы ПЛЮС.

Система PROTEE [140] автоматизирует процесс проектирования систем управления по технологической схеме, в которой указываются функции управления, информационные связи данных, а также методы контроля данных и их вычислений.

Метод проектирования основывается на предварительном анализе информации об управленческих процедурах, завершающемся описанием технологической схемы. Последняя составляется для каждой

службы предприятия. На основе этой схемы выделяются входная и выходная информация и автоматизируемые функции. Все данные (массивы, документы) описываются на специальных бланках, используемых при формировании словаря данных.

Разработка прикладных программ состоит в описании входных и выходных данных по информации, хранящейся в словаре данных, алгоритмов их обработки. Прикладные программы имеют типовую модульно-линейную организацию, включающую разделы и параграфы, в которых размещаются операторы преобразования данных средствами системы.

В идейном плане данная система и система ПЛЮС имеют много общих черт. В целом рассмотренные системы ориентированы на определенные классы предметных областей, реализуют структуры прикладных систем, которые не могут удовлетворить всему разнообразию требований, фиксируемых заказчиком в задании на разработку (быстродействие, объем памяти, необходимость работы в определенно заданной среде и т. п.).

С возникновением метода сборочного программирования появилась возможность определить **третий путь** разработки программных технологий, занимающий промежуточное положение между двумя рассмотренными. Он характеризуется тем, что вначале формируется модель разрабатываемого программного объекта, в которой специфицируются функции и проблемные данные, детализируются связи между ними. Затем определяется, подобно системе PROTEE, технологическая схема реализации модели, включающая не только процессы и действия формализованного определения элементов модели ПрО, но и требования к средствам автоматизации. Отсюда имеем прямой выход к:

определению типовых технологий (называемых в дальнейшем функционально ориентированными) для класса однотипных функций в некоторой ПрО, имеющих сходные модели программного объекта;

возможности повторного использования ранее определенных элементов функций и программ (заготовки для отдельных элементарных функций ПрО, модели программ, готовые модели и др.);

более гибкой реализации программного объекта с обеспечением требований в задании на его разработку;

специализированной инструментально-технологической поддержке процессов разработки классов однотипных программ с контрольными операциями в ходе разработки и оценки результатов труда на разных этапах жизненного цикла ППО.

Для иллюстрации применения третьего пути в качестве ПрО используется СОД, характерной особенностью которой являются большие объемы данных, требующие привлечения инструментария СУБД для их ведения и множества прикладных программ ППО, реализующих специализированные и типовые функции в различных функциональных подсистемах (МТО, ТОР и др.) и приводящих к дублированию работ и выполнению рутинных операций при сборке прикладных программ в целенаправленные комплексы и пакеты прикладных программ.

В классе указанных функциональных подсистем СОД типовыми функциями обработки информации являются:

ввод и контроль данных;

ведение данных (актуализация и загрузка) в базах данных;

обработка данных научного, исследовательского, планово-экономического и других видов;

вывод результатов обработки (в форме значений проблемных переменных, документов и т. п.) и справок о текущем состоянии баз данных.

Отмеченные функции характерны и для таких классов систем, как автоматизированные системы организационного управления, информационно-справочные системы, автоматизированные системы управления и многие другие.

Для автоматизированного создания ППО систем, реализующих приведенный набор типовых функций, при котором разработчики освобождены от многих рутинных операций программирования множества сходных по функциям прикладных программ и их сборки в программные компоненты функциональных подсистем, и поставлена задача исследования и разработки функционально-ориентированных технологий по принципу сборочного программирования.

Исходя из опыта реализации конкретных СОД, наблюдается общность и повторяемость в значительном множестве представлений отдельных функций в каждой подсистеме в виде типовых функциональных элементов разного вида. Это повлияло на развитие метода сборочного программирования для организации связей элементов повторного использования по данным и по управлению через аппарат интерфейсных функций с целью получения программных компонент функциональных подсистем СОД.

Таким образом, результаты показывают важность поставленной задачи создания технологий с функциональной ориентацией на основе метода сборочного программирования, требующего определенного развития для удовлетворения потребностей и специфических особенностей рассматриваемой предметной области СОД.

6.3. МЕТОДЫ ОПРЕДЕЛЕНИЯ ПРОЦЕССОВ И ЛИНИЙ РЕГЛАМЕНТИРОВАННОГО СОЗДАНИЯ ФУНКЦИОНАЛЬНЫХ КЛАССОВ ППО СОД

В силу большого разнообразия программных методов и средств основная задача руководства проектом состоит в том, чтобы процессы разработки удовлетворяли потребностям реализуемых функций и были по возможности автоматизированными. Поставленная задача создания функционально-ориентированной технологии (ФОТ) может быть решена посредством метода, в основе которого лежит принцип сборочного программирования с целью создания требуемой технологии из имеющихся методов и средств, максимально удовлетворяющих целям реализации проекта.

Здесь элементами сборки являются типовые программно-технологические средства и процессы на их основе, реализующие однотипные

функции ПрО и оснащенные известными или оригинальными программными методами и средствами. Вопросам определения процессов проектирования программных систем посвящено большое количество публикаций [10, 17, 18, 24, 32, 65, 66, 97, 104, 108, 111, 144 и др.]

Все работы в той или иной степени отталкиваются от модели жизненного цикла, адаптируя ее к рассматриваемому классу ПрО и используя программные методы и средства для последовательного перехода одного состояния объекта в другое.

Сложность объекта является главным фактором, влияющим на состав технологических процессов в ТЛ. В качестве примера можно привести программный объект — пакет прикладных программ статистики, реализованный в рамках ППО СОД, ТЛ разработки которого включает 13 технологических процессов. Каждый этап модели жизненного цикла данного программного объекта оказался представленным более чем одним процессом. В целом совокупность процессов этапа предназначается для получения нового состояния объекта из отдельных составляющих каждого процесса.

Принципы построения ТЛ. Процесс построения технологии прикладного программирования, ориентированный на реализацию функций ПрО СОД, является трудоемким и творческим. Он выполняется высококвалифицированными специалистами (функциональщиками, аналитиками, системными программистами и др.), знающими реализуемую ПрО и технологические аспекты создания современных прикладных систем. ФОТ отображается в ТЛ и задает среду прикладного программирования, которая методически, программно и организационно поддерживается специальными программно-технологическими компонентами — технологическими модулями.

Описание ТЛ должно ориентироваться на эффективное управление разработкой, включающее планирование сроков и трудоемкости, постепенное повышение производительности труда исполнителей за счет создания фондов технологии (как готовых повторно используемых программных элементов, так и типовых технологических процессов). Входящие в ТЛ процессы должны обеспечивать изменение состояния объекта и оценку показателей качества в процессе разработки объекта.

Принятие технологических решений по составу и структуре ТЛ основывается на результатах анализа ПрО, сопоставлении со свойствами имеющихся аналогов и всестороннего анализа требований заказчика к разрабатываемому программному объекту.

Выделение ТП в ТЛ представляет собой многошаговый итерационный процесс, протекающий в двух (встречных) направлениях; первоначально весь жизненный цикл объекта делится на крупные этапы (см. гл. 1), затем в рамках каждого этапа происходит выделение более низких уровней (с различной степенью детализации) и, наконец, выполняется группирование ранее выделенных работ по ТП, при этом возможны повторное расчленение некоторых работ и их перегруппирование по ТЛ.

После определения структурного и функционального назначения и способов взаимосвязи выделенных ТП осуществляется анализ имею-

щихся типовых ТП с целью их включения в разрабатываемую ТЛ. Если такие ТП имеются в технологическом фонде, то проводится их адаптация к условиям применения в данной ТЛ. Адаптация предполагает добавление новых технологических операций, незначительное переформулирование существующих операций в ТП, уточнение состава и функций проектно-технологических документов (ПТД), применяемых для фиксации результатов разработки на ТП.

Для каждого ТП определяется организационная структура коллектива, выполняющего разработку по ТП, с учетом профессиональной специализации и квалификационного уровня исполнителей каждой операции. Рекомендуемая на ТЛ организационная структура (фактически модель коллектива разработчиков) уточняется при настройке ТЛ на конкретный коллектив и средства разработки.

Технологические операции и технологические процессы объединяются во взаимосвязанную совокупность с последовательным, параллельным, циклическим или условным режимом выполнения посредством технологического маршрута.

Формализация описания ТЛ. Для описания конкретной функционально-ориентированной ТЛ могут быть использованы язык спецификаций табличного вида и полуформальное описание семантики ТП и ТЛ в виде комплекта технологических документов.

Язык спецификаций включает табличные формы, в которые заносится информация об операциях ТП и процессах ТЛ. Каждая строка таблицы ТП, называемая картой ТП (КТП), специфицирует операцию процесса, входные и выходные данные, представляемые в ПТД, метод, средство и инструмент выполнения операции, категорию исполнителя, метод контроля и оценки результата процесса.

Поскольку основным назначением процесса является задание среды изменения состояния модели программы на этапе (процессе) жизненного цикла, то входные и выходные данные процесса — суть этого состояния.

Спецификация выделенных процессов задается в КТП и карте ТЛ, каждая строка которой содержит описание процесса, входных и выходных его данных, задаваемых в ПТД. Для каждого ТП и ТЛ в целом строятся технологические маршруты, специфицирующие связи между ТП и их операциями. Набор КТП, технологических маршрутов ТП и ТЛ образует полную спецификацию ТЛ. В описание ТЛ входит также описание методик и инструкций, а также правил применения ТМ в автоматизированном режиме.

Выбранная форма спецификаций табличного вида и технологические маршруты как раз позволяют провести автоматизированное управление ходом выполнения операций и оценку их результатов. Поскольку каждая операция (в соответствии с требованиями к ТП) поддерживается технологическим модулем, то процесс управления и выполнения его осуществляет компонента, называемая технологическим диспетчером.

Модель среды разработки программного объекта по ТЛ представлена на рис. 6.2, а модель отдельного процесса — на рис. 6.3.

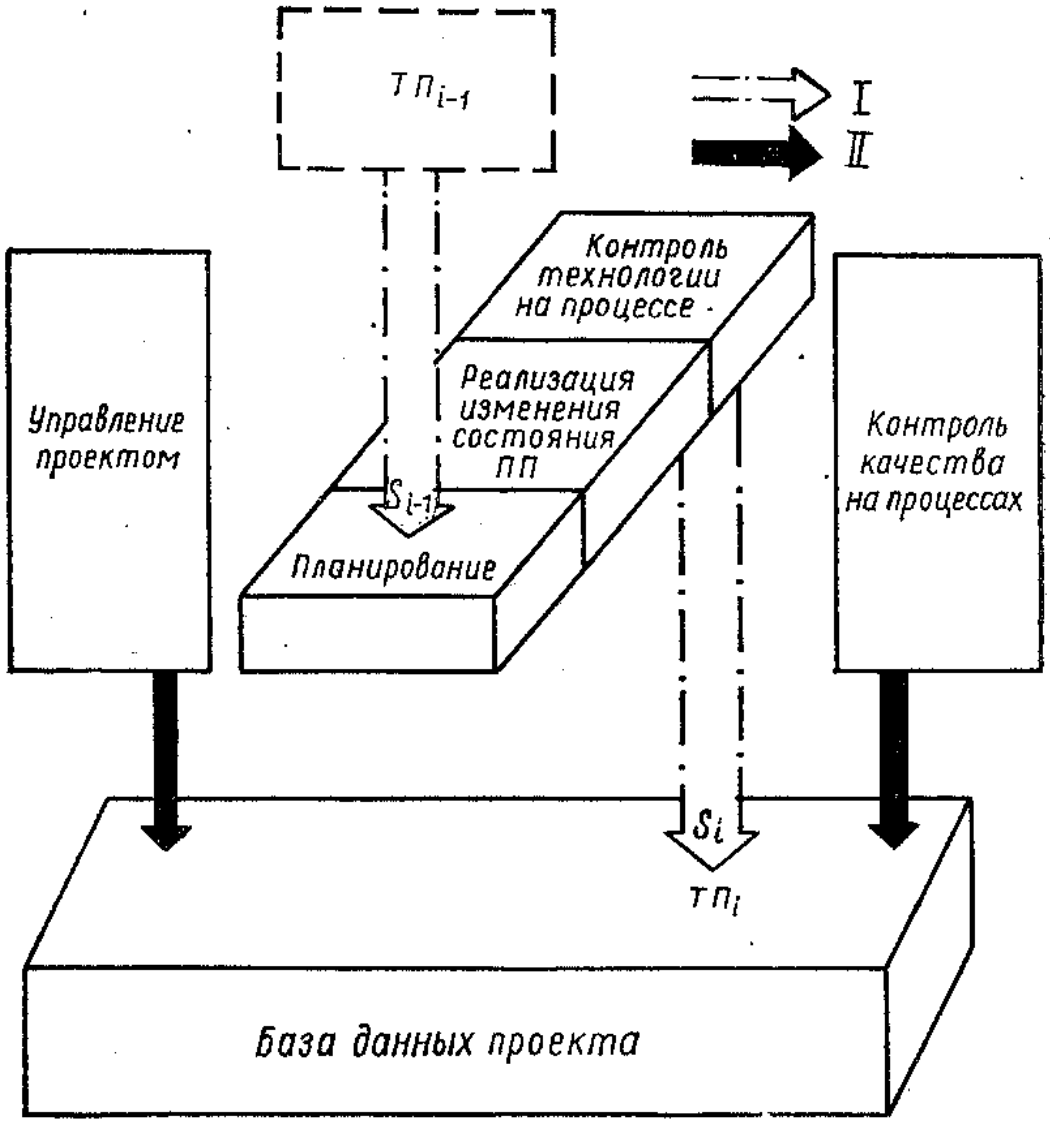


Рис. 6.3. Модель среды технологического процесса:
I — состояние программного продукта на ТП$_i$ (S_i); II—проектные решения по качеству ПП; Si — преобразованная функциональная архитектура.

Инструментальным средством автоматизации ТЛ является программно-технологический комплекс (рис. 6.5). Функцию управления процессом проектирования прикладных программ на основе ТЛ осуществляет монитор. Он использует информационную базу комплекса (ИБК), в которой хранятся все виды моделей, соответствующие ТЛ$_1$, ..., ТЛ$_N$. Результаты проектирования запоминаются в информационной базе данных проекта (БДП).

6.4. ОСНОВНЫЕ МОДЕЛИ СБОРКИ ФУНКЦИОНАЛЬНО-ОРИЕНТИРОВАННЫХ ТЕХНОЛОГИЙ

Для проектирования и разработки методом сборки новых функционально-ориентированных технологий (ФОТ) используется класс взаимосвязанных моделей, позволяющих описать с разных позиций взгляд специалиста предметной области на способы представления ФОТ.

Класс моделей включает модели формализованного представления процессов и линий разработки, состояний программ, отображаемых в ходе разработки моделями следующего вида:

1) модель качества ПС ($M_{кач}$);

2) систему моделей (бланков) формализованного представления проектных решений в ходе разработки ($M_{пдт}$);

3) модель эксплуатационно-программных документов ($M_{эпд}$);

4) модели формализованного представления спецификаций технологических процессов и линий ($M_{тп}$, $M_{тл}$).

Определим эти модели.

Модель качества $M_{кач}$. Качество ПС характеризует его пригодность к использованию по назначению. Для разных типов ПС в классе задач ПрО предварительно разрабатывается номенклатура показателей и устанавливаются их базовые значения, которые должны быть достигнуты при разработке отдельных прикладных программ по их моделям $M_{пп}$.

Модель $M_{кач}$ — это формализованное представление пользовательских требований к свойствам ПС и способов их оценки на этапах жизненного цикла, отраженных в ТЛ.

Данная модель $M_{кач}$ является четырехуровневой (рис. 6.4). На первом находятся факторы качества, отображающие потребительские свойства, присущие созданной ПС.

Показатели второго уровня — это комплексные показатели (корректность, надежность, переносимость и др.), необходимые для разрабатываемой ПС, чтобы достигнуть заданные свойства качества первого уровня. Каждый комплексный показатель представляется набором единичных признаков и соотносится с соответствующим свойством качества первого уровня, т. е. каждый показатель определяется набором отдельных его признаков, которые необходимо достигнуть в процессе разработки ПС на этапах жизненного цикла и использовать их при комплексной оценке.

На третьем уровне в иерархической модели оценки качества находятся так называемые метрики. Метрика — это совокупность оценочных элементов, позволяющих определить степень достижения признаков комплексных показателей качества. Метрика, характеризующая показатель качества, может иметь один или несколько оценочных элементов.

Процесс метрического оценивания осуществляется в ходе разработки программного продукта на этапах жизненного цикла ТЛ:

путем экспертизы состояний программного объекта (структуры программы, правильности оформления документации и др.) и заполнения протокола (карты) экспертизы;

аналитическим путем на основе информации, формируемой в картах регистрации ошибок, о результатах тестирования объекта на соответствующих процессах ТЛ посредством расчета оценки надежности соответствующими математическими методами расчета.

На четвертом уровне находятся оценочные элементы, являющиеся элементарными характеристиками отдельных свойств создаваемого

Факторы свойств	Комплексные показатели свойств	Оценочные элементы показателей	Метрики оценочных элементов	Способы оценки
Назначение	Сложность	Неупорядоченность	Экспертиза структуры системы, компонент, объема	Методы оценки сложности (гл. 8)
		Связанность		
		Объем		
	Функциональность	Коэффициент полноты реализации функций	Экспертиза полноты реализации функций	1 — функция реализована 0 — нет
	Защищенность	Коэффициент защищенности	Экспертиза средств защиты	$0{,}75 \leqslant K_з \leqslant 1$
Функционирование	Корректность	Непротиворечивость	Экспертиза непротиворечивости, согласованности, помехоустойчивости	0 или 1 расчет по формуле (гл. 8)
		Согласованность		
		Количество оставшихся ошибок		
	Точность	Погрешность вычислений	Точность вычислений	0,9 | 1
	Надежность	Безотказность	Экспертиза безошибочности функционирования и восстановления	Оценочные методы надежности (H) $0{,}75 \leqslant H \leqslant 1$
		Восстанавливаемость		
		Коэффициент готовности		
	Эффективность	Быстродействие	Экспертиза обеспечения эффективности	0 или 1
		Экономия памяти		

Эргономичность	Простота	Количество операторов по подготовке к работе	Экспертиза простоты обучения и работы	Оценка по формулам (гл. 8)
	Подготовка к работе	Доля операторов с подсказкой		0 или 1
	Обучаемость	Наличие обучающих курсов		
	Переносимость	Доля адаптации к другой среде Степень применимости ЯВУ	Экспертиза независимости от техники и среды	0 или 1
	Документируемость	Полнота ЭПД Простота изложения материала	Экспертиза ЭПД, легкости освоения	0 или 1
Технологичность	Разработки	Структурируемость Степень соблюдения технологии Трудоемкость разработки	Экспертиза структурности, интерфейсов, технологии разработки	0 или 1
	Внедрения	Трудоемкость внедрения	Оценка трудоемкости внедрения и сопровождения	Оценка по формулам (гл. 8)
	Сопровождения	Трудоемкость сопровождения Степень использования ЯВУ		
	Стандартизация	Коэффициент повторяемости, стандартности	Экспертиза использования стандартов, коэффициентов повторной используемости (КП), унифицированности	Оценка по формулам (гл. 8)
	Унификация	Коэффициент унификации		

Рис. 6.4. Модель качества ППО СОД

ПС. Набор оценочных элементов группируется и распределяется по соответствующим этапам (процесссам) технологии разработки ПС.

Текущая оценка отдельных элементов качества проводится на всех ТП. Контрольными точками проверки являются специальные технологические операции, выполняемые группой инспекции результатов.

Комплексная оценка качества созданной ПС проводится исходя из промежуточных значений оценочных элементов, распределенных по ТП.

Модели $M_{\text{птд}}$ формализованного представления проектных решений в ходе разработки образуют набор моделей, позволяющих осуществлять выполнение следующих функций разработчика в процессе разработки:

описания результатов проектных решений в ПТД табличной формы, принятой в ТПР и в конкретных технологических линиях;

проведения контроля выполнения преобразования состояния программного объекта и фиксации ошибки;

занесения фрагментов текстов для включения в соответствующие разделы документов программной документации согласно модели $M_{\text{эпд}}$.

Выбор формы представления моделей ПТД для конкретной ТЛ зависит от класса реализуемых объектов. Как показал опыт разработки, при проектировании ПС, работающих с БД, используются модели типа «сущность — связь», для которых применяются специальные формы ПТД, более полно отображающие специфику программ их реализации. Наиболее часто используемой формой стандартного типа для многих классов создаваемых программ являются ПТД, приведенные в табл. 6.1 и 6.2. В них описываются структура данных и логика алгоритма программ.

Таблица 6.1

ТО	Таблица данных						Лист	
Порядковый номер	Уровень	Идентификатор	Тип	Шаблон	Диапазон	Точность	Начальное значение	Примечание

Модели $M_{\text{тд}}$, $M_{\text{тп}}$ формализованного представления технологических процессов и линий, обеспечивающих построение любых видов программ из заданного класса.

Модель процесса (линии) — это типовая структура (карта) процесса (линии) для формализованного задания последовательности операций (процессов), связь между которыми задается технологическим маршрутом, отображая логико-функциональную взаимозависимость операций (процессов).

Таблица 6.2

ПМ	Логика алгоритма	Лист	Листов

1. Назначение

2. Среда

ЯП	ОС	Хранится	Память	Время	Системное ПО

3. Использует

4. Формирует

5. Тест

Модели $M_{тл}$ и $M_{тп}$ служат способом спецификации формируемых функционально-ориентированных технологий и моделирования этапов жизненного цикла создаваемого программного объекта.

$M_{тп}$ определим на технологических объектах:

$$M_{тп} = \{КТП,\ ТМШ,\ M_{птд},\ ТД_{тп}\},$$

где КТП — структура карты процесса, принятая в ТПР (табл. 6.3) для фиксации операций процесса; ТМШ — графовая форма представления технологического маршрута; $M_{птд}$ — зафиксированные для данного ТП из множества ПТД модели табличных документов, используемые при описании состояния программного объекта на этом процессе; $ТД_{тп}$ — структура технологических документов ТП для описания прагматики и семантики процесса ведения разработки соответствующего состояния объекта.

Используя структуру описания отдельного процесса, можно представить модель ТЛ следующим образом:

$$M_{тд} = \{\{M^i_{тп}\}\ ТМШ_{тп}\}\ i = \overline{1, N}.$$

Здесь элементами модели являются наборы моделей технологических процессов и модель технологического маршрута линии. Модель ТЛ — абстрактная модель управления процессом разработки ПС на инженерной основе с обратными связями и дискретным характером взаимодействия между каждым ТП. ТЛ содержит управляющую часть, представленную технологическим маршрутом, задающим связи между процессами, и операционную, задаваемую картами ТП в виде набора ТО.

Разделение ТЛ на управляющие и операционные элементы способствует ее настройке на реальные условия,заключающиеся в конкретизации круга специалистов, уточнении документов и инструментов и составлении сетевого графика работ в соответствии с технологическим маршрутом.

Таблица 6.3

КТП	Карта технологического процесса							Лист	Листов
Код ТП	Наименование ТП								
Код ТО	Наименование (краткое содержание) ТО	ПТД				Исполнитель	Методы и средства выполнения ТО		Используемые документы
		исходные		выходные			технологические	инструментальные	
		Код	Наименование	Код	Наименование				
1	2	3	4	5	6	7	8	9	10

Разработал	ФИО	Подпись	Дата
Принял			
Утвердил			

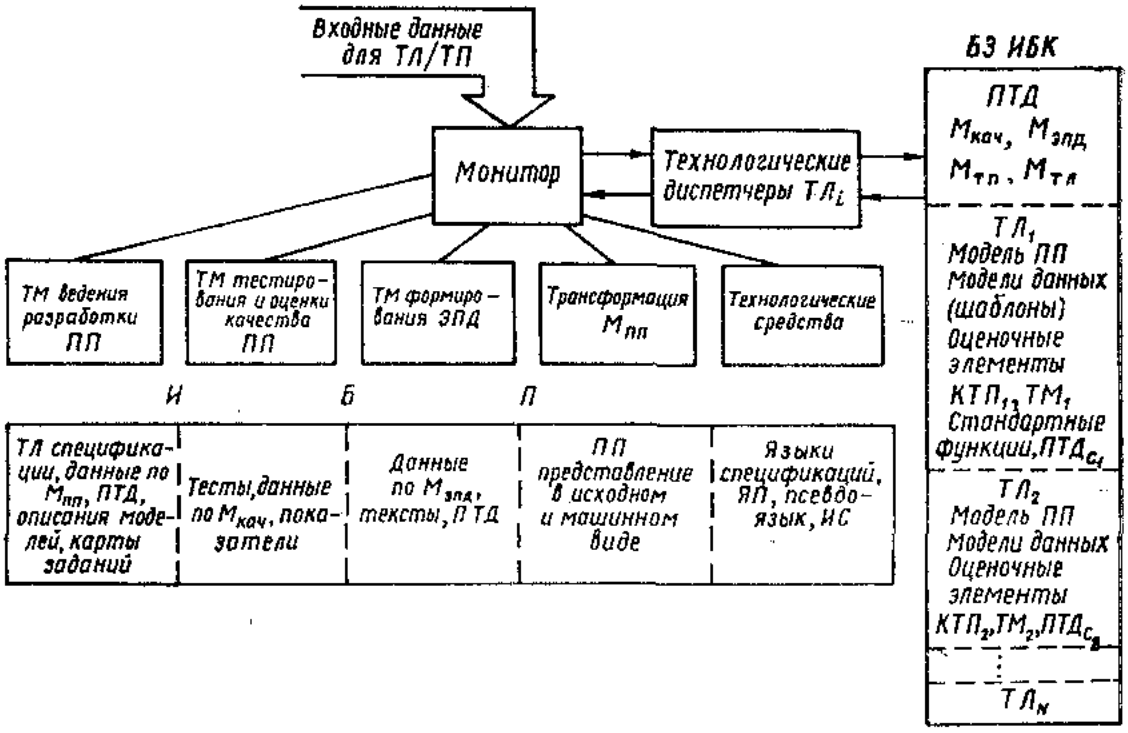


Рис. 6.5. Структура программно-технологического комплекса

Данное представление ТЛ ассоциируется с характером современных гибких автоматизированных линий приема, передачи и обработки информации в распределенных системах обработки информации (рис. 6.5).

Модель ЭПД является формализованным представлением принятой в ЕСПД структуры каждого типа программной документации (ГОСТ 19.501—19.507—77), полученным исходя из анализа требований ГОСТов и специфики представления отдельных функций ПрО набором функционально-ориентированных заготовок. Для них делается описание, которое затем используется при выводе программы из ее модели по заготовкам. Эти описания размещаются в соответствующих местах шаблонов документов.

Другим источником информации для формирования документации по $М_{эпд}$ являются ПТД, заполняемые в ходе разработки. Модель $М_{эпд}$ включает набор подмоделей (структур) конкретных документов ЭПД и базовых компонентов ($М_{бк}$), содержащих формальное описание общих для всех ЭПД элементов, т. е.

$$М_{эпд} = \{М_{бк},\ М_{оп},\ М_{рпр},\ М_{рсрп},\ М_{ро}\},$$

где $М_{оп}$ — модель описания применения; $М_{рпр}$ — модель руководства программиста; $М_{рсрп}$ — модель руководства системного программиста; $М_{ро}$ — модель руководства оператора и др.

Представленный класс технологических моделей не является замкнутым и может пополняться новыми моделями, учитывающими специфику разрабатываемого объекта. Состав и структура технологических моделей, используемых при разработке ПП, должны определяться на этапе ТПР программных систем соответствующего класса и служить основой реализации ТЛ. Разработчику ТЛ представляется пра-

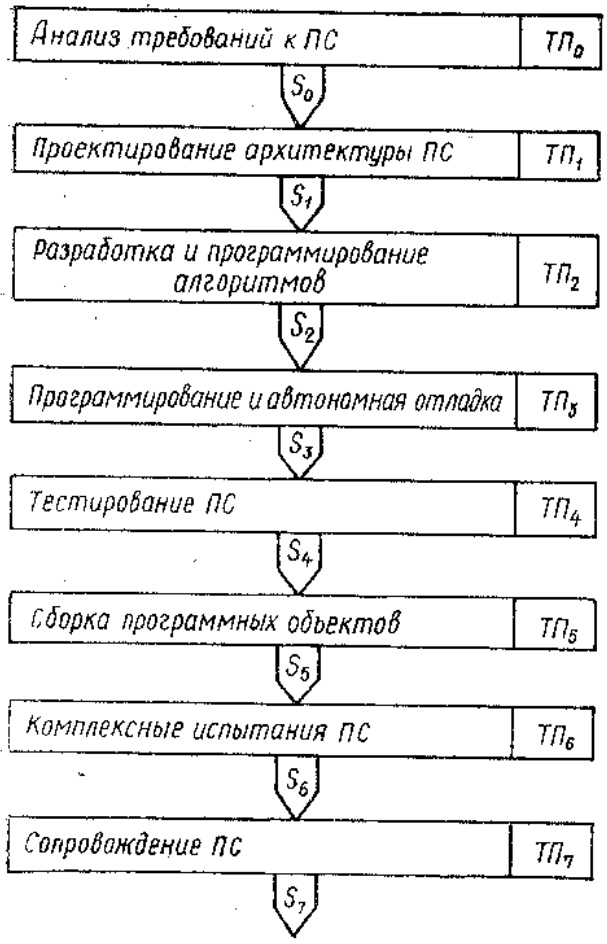


Рис. 6.6 Модель жизненного цикла ПССОД

во выбора любых доступных средств формализации (табличная модель, сетевая модель и др.) элементов ПП.

Модель жизненного цикла ПС. Процесс разработки ПС определяется моделью жизненного цикла, этапы которой будем отождествлять с технологическими процессами. Для каждого ТП фиксируются начальное (S_0), промежуточное (S_i) и конечное (S_k) состояния разрабатываемого программного объекта. Переход объекта из состояния S_i в S_{i+1} на $ТП_i$ связан с выполнением технологических операций, входящих в этот ТП. При этом для каждого типа ПП из заданного класса программных систем может быть выбран свой набор ТП и состояний ТП, определяемых на соответствующем множестве исходных данных.

Приведенная на рис. 6.6 схема проектирования соответствует методу проектирования ПС сверху-вниз, при котором понятия $(i + 1)$-го уровня ($ТП_{i+1}$) описываются через элементарные понятия i-го уровня и представляют собой данные, входящие в класс понятий состояния ПС. При этом не исключаются переходы с одного уровня абстракции на другой (сверху-вниз и снизу-вверх).

Модель процесса разработки. Выделение состава $ТП_i$ и содержания технологических операций зависит от класса ПС, используемой методологии и принятой организации проведения работ. При определении состава и содержания ТП решается задача обеспечения инженерной дисциплины разработки, основанной на современных программных принципах, методах и инструментах, способствующих полному и адекватному отображению понятий и функций реализуемой предметной области в элементы состояния S. Основная цель выделяемых $ТП_i \subset ТЛ$ состоит в получении некоторого полуфабриката (S_i — состояние ПС), фрагментов ЭПД и в проведении экспертизы научно-технического уровня и качества разработки в соответствии с моделью качества ($M_{кач}$), разработанной на этапе анализа требований к ПС. Каждый ТП модели жизненного цикла в общем виде определяет состояние элементов ПС, состав технологических операций, обеспечивающих преобразование исходного состояния ПС и получение его конечного состояния.

Таким образом, общая технологическая модель (схема) процесса разработки является отражением модели жизненного цикла, способов преобразования ПС и может быть представлена в виде

$$M_{\text{пр}} = \{S, \text{ТП}_i(\text{ТО}_j), \text{ТМ}_j\}, \; i = \overline{0, 7}, \; j = 1, k. \qquad (2.6)$$

Множество состояний S рассматриваемой модели включает:

S_0 — исходное (начальное) состояние — описание требований заказчика к данному ПС;

S_1 — состояние, включающее набор элементарных состояний — описаний функций (S_1^1), архитектуры (S_1^2), структуры данных (S_1^3) и т. п. средствами класса языков спецификаций L_1;

S_2 — состояние соответствует техническому проекту и включает описание в классе языков L_2 алгоритмов функции (S_2^1), данных (S_2^2), интерфейсов (S_2^3), гипертекстов документации (S_2^4), оценочных элементов модели качества (S_2^5) и др.;

S_3 — состояние соответствует рабочему проекту и включает описание в языках программирования (класс L_3) текстов программ (S_3^1), модулей (S_3^2), тестов (S_3^3) и т. п.;

S_4 — состояние соответствует отлаженным элементам программного продукта;

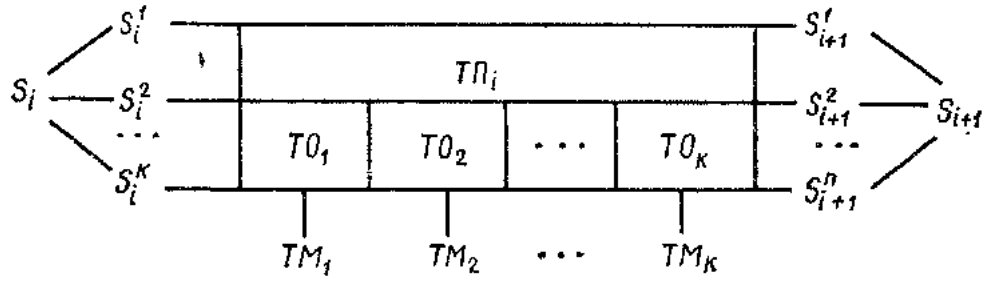
S_5 — состояние — это ПС после сборки;

S_6 — состояние ПС, соответствующее программному продукту, которое испытывается, проверяется на соответствие заданным функциям и значениям показателей качества;

S_7 — состояние ПС на этапе сопровождения.

Технологический процесс, входящий в состав $M_{\text{пр}}$, является промежуточным (частичным) процессом, осуществляющим преобразование S_i-го состояния ПС с помощью набора $\text{ТО}_j \subset \text{ТП}_i$ данной $M_{\text{пр}}$, каждая из которых поддерживается ТМ_j (либо один ТМ реализует несколько ТО).

Набор операций $M_{\text{пр}}$ не является фиксированным, он уточняется для каждого типа ПС и описывается в технологических документах. Среди операций могут быть типовые, используемые готовыми при создании конкретной технологии разработки ПС определенного типа. Для обеспечения технологичности разработки в их состав входят операции управления качеством разработки и формирования документации на всех этапах жизненного цикла в соответствии с системой моделей $M_{\text{эпд}}$, определенной выше.



6.7. Модель частичного ТП_i процесса

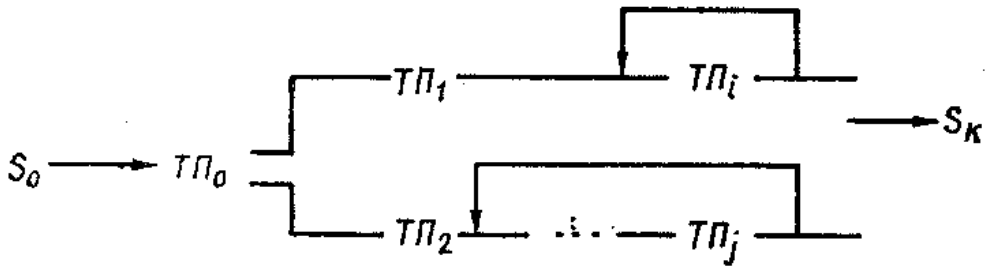


Рис. 6.8. Графовая модель технологического процесса

Модель любого частичного $ТП_i$ в общем виде представлена на рис. 6.7.

Состояние объекта S_i определяется набором частичных его состояний $S_i^1, \ldots, S_i^k$, подаваемых на вход $ТП_i = \{ТО_k\}$. Данный набор для конкретного ПС конечен.

Управление процессом преобразования состояний объекта S_0 в S_k может быть задано в виде графовой модели (что соответствует технологическому маршруту), в вершинах которой находятся процессы (или операции), а ребра задают всевозможные переходы из одного состояния объекта в другое. Графовая модель отображает способ ведения разработки с распараллеливанием работ и возвратами (при ошибках) в предыдущие процессы (рис. 6.8).

Каждый частичный процесс $ТП_i$ является абстрактным конечным автоматом, граф которого совпадает с технологическим маршрутом процесса, множество состояний S_i которого определено на совокупности операций $\{ТО_i\} \subset ТО$ данного процесса.

Переход объекта из состояния S_i в состояние S_{i+1} может быть только под действием элементов технологического маршрута и осуществляется технологическим диспетчером посредством выбора из множества операций процесса, зафиксированных в карте i-го процесса, очередной операции при условии, что результат предыдущей операции проанализирован и выполнен.

Задача выполнения i-го процесса заключается в переводе автомата из некоторого промежуточного состояния объекта в другое S_{i+1} с выполнением операций преобразования, качественной или количественной оценки результата разработки объекта и формирования фрагментов ЭПД по $M_{эпд}$.

Глава 7

СРЕДСТВА РАЗРАБОТКИ ПС НА ОСНОВЕ ТЕХНОЛОГИЧЕСКИХ ЛИНИЙ

В настоящей главе описаны средства разработки ПС с позиции понятия технологической линии в рамках технологии сборочного программирования. Данные средства рассматриваются как абстрактные системы. Цель изложения — рассмотреть типовые процессы и операции разработки ПС на основе применения метода сборочного программирования. Приводятся примеры программных средств реализации этих операций и процессов. Несмотря на определенный уровень абстракции, для рассматриваемых ПС имеются реальные прототипы, разработанные в Институте кибернетики имени В. М. Глушкова АН УССР.

Средства поддержки любой технологии программирования предполагают наличие инструментальных ПС и правил их применения. Конкретные инструментальные средства совместно с соответствующими процедурами их использования будем называть технологическим модулем (ТМ). В качестве примеров рассмотрены средства сборки программ из модулей и проектирования методо-ориентированных ППП. Для них введем условные обозначения ССПМ (прототип — система АПРОП [30, 41]) и СППО (прототип — комплекс АПФОРС [101]) соответственно.

7.1. ТЕХНОЛОГИЧЕСКИЙ МОДУЛЬ ССПМ

Рассматриваемый ТМ предназначен для выполнения:

сборки, тестирования и отладки отдельных программных компонентов (ПК) из модулей, написанных на различных языках программирования и автономно отлаженных вне среды данного ТМ;

сборки, тестирования и отладки программного продукта из программных компонентов, собранных и отлаженных в среде данного ТМ;

получения копий собранного ПП (или его части) в хранилищах пользователя ТМ в виде загрузочного агрегата, готового к функционированию в среде ОС.

Инструментальной основой ТМ является ССПМ, предназначенная для автоматизированного изготовления ПП различного уровня слож-

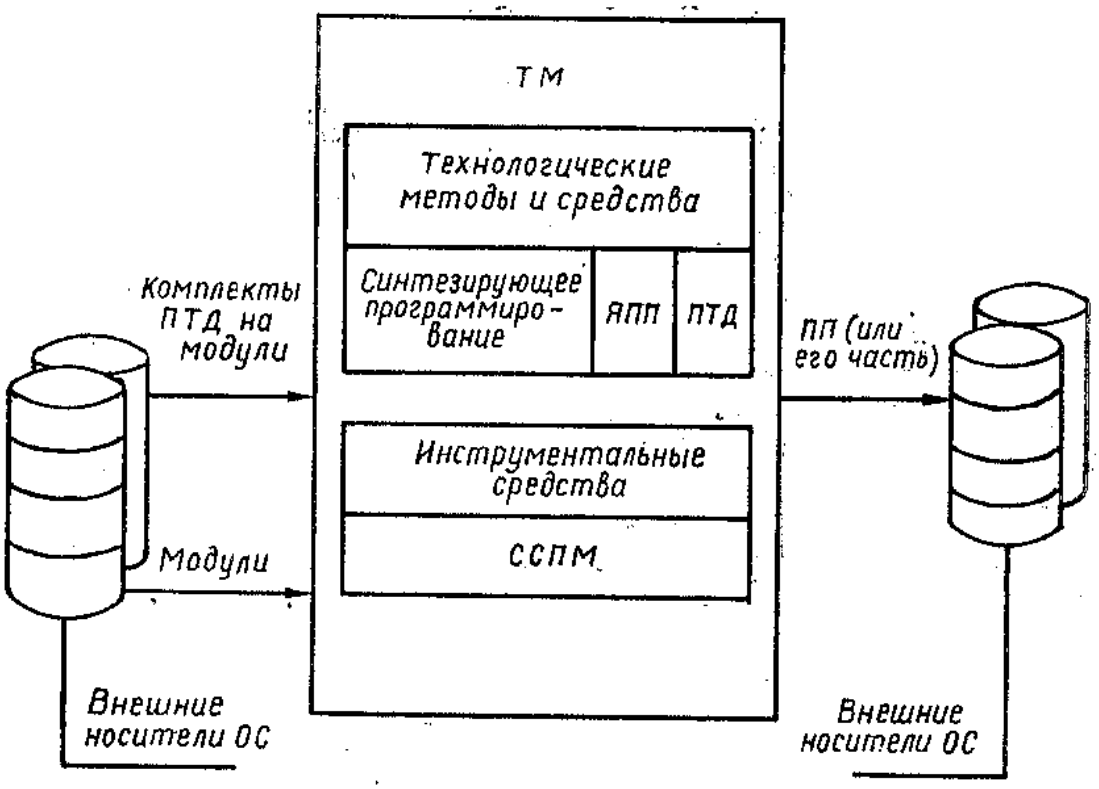


Рис. 7.1. Состав технологического модуля сборки, тестирования и отладки ПП

ности на основе методологии модульного программирования. Процесс конструирования ПП осуществляется в диалоговом режиме с использованием языка модульного конструирования (ЯМК).

Средства ССПМ обеспечивают:

ввод исходных текстов программных объектов в базу данных как с терминалов, так и с внешних носителей (вводимый объект сопровождается паспортом, содержащим сведения о его назначении, входных и выходных параметрах, вызываемых объектах, требуемых программно-информационных ресурсах и т. д.);

ведение базы данных объектов (создание, уничтожение, занесение, корректировка и удаление из них различных объектов);

обработку объектов с использованием средств ОС (трансляция, редактирование, счет и т. п.);

сборку одноязыковых и разноязыковых модулей в программный агрегат;

отладку агрегата;

отторжение агрегата в загрузочном виде от системы (перенос в хранилища ОС).

Доступ к указанным возможностям осуществляется в режиме мультиобработки, позволяющем одновременно работать с системой нескольким пользователям.

Технологическую основу ТМ составляют методы и средства, поддерживающие стратегию синтезирующего (восходящего) программирования с соблюдением принципа модульности. В качестве базового технологического средства используется ЯМК, на котором описывается входная информация об объектах сборки (ее состав регламенти-

Таблица 7.1

Номер операции	Операция	Оператор ЯМК (в реальной системе)	Ссылка на документы
1	Подготовка исходной информации		Технология разработки ПС
1.1	Подготовка проектно-технологических документов	—	То же Методика применения технологических модулей разработки ПС
1.2	Подготовка исходных текстов модулей	—	ГОСТ 19
1.3	Подготовка ССПМ к запуску	—	Руководство программиста
1.4	Запуск ССПМ и начало работы	SUPER	То же
2	Погружение модулей и форм проектных документов в среду ССПМ	SUBSYS	Технологическая инструкция
3	Сборка, тестирование и отладка ПП	PKD	Методики сборки, тестирования и отладки
3.1	Просмотр и редактирование паспортов и исходных текстов модулей	CORR TLIB	Руководство программиста
3.2	Создание личной библиотеки (ЛБ)	CREATE	То же
3.3	Запись исходного текста модуля в ЛБ	LOAD	»
3.4	Описание исходных модулей, находящихся в ЛБ	DSCR	»
3.5	Трансляция модуля	TRANS	»
3.6	Просмотр и редактирование паспортов и исходных текстов модулей из ЛБ	CORR PLIB	»
3.7	Занесение объектного вида модуля в ЛБ	LOAD	»
3.8	Описание объектного вида модуля из ЛБ	DSCR	»
3.9	Построение графа ПП	LINK	»
3.10	Отладка ПП со включением средств трассировки передач управления и параметров	LINK	»
3.11	Сборка ПП без включения средств трассировки	LINK	»
3.12	Подготовка тестовых данных	CORR	»
3.13	Тестирование ПП на тестовых данных	EXEC	»
3.14	Просмотр результатов выполнения	LIST	»
3.15	Распечатка результатов или их фрагментов	PRINT	»
3.16	Запись созданного ПП и ЛБ	LOAD	»

Продолжение табл. 7.1

Номер операции	Операция	Оператор ЯМК (в реальной системе)	Ссылка на документы
4	Перенос ПП или его части в среду ОС и завершение работы		
4.1	Запись созданного ПП в библиотеку загрузочных модулей ОС	SUBSYS OS	Технологическая инструкция
4.2	Завершение работы системы, получение протокола и результатов работы системы	SEND или CANCEL	Руководство программиста
4.3	Анализ протокола работы системы и результатов тестирования ПП	—	—

руется формами проектно-технологических документов, ранее приведенных в табл. (6.1—6.9), автоматически преобразуемая впоследствии в паспорт объекта для ССПМ. Эта информация подается на вход ССПМ с внешних файлов. Схематически состав ТМ представлен на рис. 7.1.

7.1.1. ПРИМЕНЕНИЕ ТМ ДЛЯ СБОРКИ, ТЕСТИРОВАНИЯ И ОТЛАДКИ ПП

Применение ТМ в целях сборки, тестирования и отладки ПП состоит в выполнении следующей последовательности технологических операций (ТО):

подготовка исходной информации;

погружение описаний модулей и соответствующих проектно-технологических документов (ПТД) в среду ССПМ;

сборка, тестирования и отладки ПП (или его части) средствами ССПМ;

перенос ПП (или его части) из среды ССПМ в среду ОС и завершение работы.

Каждая из этих ТО предусматривает выполнение последовательности действий, представленных в табл. 7.1.

7.1.2. ПОДГОТОВКА ИСХОДНОЙ ИНФОРМАЦИИ

Подготовка ПТД. Они разрабатываются на этапе проектирования и заполняются машинным способом на бланках ПТД средствами любого редактора текстов. Затем объединяются в виде групп записей в один (или несколько) последовательный файл на внешнем носителе, доступ к которому возможен средствами ОС.

В состав комплекта ПТД на модуль входят паспорт модуля, описание параметров и описание логики. ПТД оформляются в соответствии с требованиями ССПМ. На основе этих документов производится автоматическая генерация паспортов модулей в системном виде.

При описании паспорта модуля указываются имя модуля и список формальных параметров. Входные параметры этого списка отделяются от выходных точкой с запятой. Затем размещаются дополнительные разделы: ПАРАМЕТРЫ и ВЫЗЫВАЕМЫЕ МОДУЛИ. Они содержат описание параметров, следующих за ключевым словом ПАРАМЕТРЫ, и список вызываемых модулей, указываемый после ключевых слов ВЫЗЫВАЕМЫЕ МОДУЛИ.

Если формат параметра не указан, то используется принцип умолчания, по которому определяется значение, соответствующее его типу и коду ЯП. Задание фактических параметров, передаваемых вызываемому модулю, обязательно только в том случае, когда предполагается сборка разноязыковых модулем. В случае связи одноязыковых модулей указывается список имен вызываемых модулей, которого достаточно для тестирования связей по управлению.

При использовании таблицы данных (см. табл. 6.1) описание параметров выполняется на ЯМК. Обязательными являются поля «уровень», «имя», «размерность» и «тип». Если не заполнено поле «формат», принимается значение по умолчанию.

Подготовка исходных текстов модулей. Кодирование модулей выполняется с помощью ПТД средствами ЯП. Тексты модулей оформляются в соответствии требованиям ГОСТ 19401—78 и ЯП. Правильные тексты (после трансляции) сохраняются в библиотеке исходных текстов для последующего их использования при сборке и тестировании в среде ССПМ.

Подготовка ССПМ и запуск. Задание на запуск вводится из файла или с дисплея. Подготовка системы к запуску состоит в настройке задания на конкретную конфигурацию технических средств и в подключении требуемых наборов данных. Такими могут быть: наборы данных, содержащие ПТД; библиотечные наборы данных ОС, содержащие исходные тексты модулей; наборы данных, используемые при работе ПП (в том числе описывающие входные данные).

Запуск ССПМ осуществляется средствами ОС. При этом возможен ряд нерегулярных ситуаций, связанных с техническим состоянием терминалов или их занятостью, отсутствием свободной памяти под временные и динамически заказываемые системой наборы данных. После этого система выделяет терминалы, с которых вводятся: режим работы (диалоговый или пакетный); различные параметры функционирования; операторы ЯМК.

Таким образом, пользователь осуществляет сеанс работы в системе. В дальнейшем он может завершить сеанс и начать новый либо отключить терминал от системы.

7.1.3. ПОГРУЖЕНИЕ МОДУЛЕЙ И ПТД В СРЕДУ ССПМ

Эти действия выполняются с помощью оператора SUBSYS в форме диалога с пользователем. По запросам пользователя данный оператор осуществляет: подключение к системе библиотек исходных модулей

(БИМ) ОС, содержащих модули для сборки и тестовые модули, а также ввод этих модулей во временную библиотеку монитора (ВБМ) ССПМ; подключение к системе файлов ОС, содержащих ПТД на модули, участвующие в сборке; автоматическую генерацию паспортов этих модулей во временной библиотеке паспортов (ВБП).

Сведения о модулях, введенных в ВБМ, пользователь может получить, введя со своего экрана оператор описания модулей. По этому оператору на экран выдается таблица, содержащая сведения об именах, видах, местонахождении и языке объектов, с которыми работают пользователи.

Исправления в текстах погруженных в среду ССПМ, исходных модулей и паспортов выполняются с помощью операторов ЯМК:

просмотр и редактирование паспорта модуля

CORR PAS TLIB ⟨имя модуля⟩;

просмотр и редактирование текста модуля;

CORR TLIB ⟨имя модуля⟩.

Просмотр и редактирование выполняются в ВБМ, существующих только на время сеанса работы.

Создание личной библиотеки. Модули разрабатываемого ПП целесообразно хранить в постоянных архивах личных библиотек (ЛБ). Их создание выполняется оператором

CREATE PLIB ⟨имя ЛБ⟩.

При этом на соответствующем томе системы должен быть достаточный объем памяти под ЛБ, уточняемый системой в диалоге с пользователем.

Запись модуля в ЛБ. Выполняется с помощью оператора

LOAD MOD ⟨имя ЛБ⟩. ⟨имя модуля⟩ SOUR.

Результатом работы является занесение модуля в ЛБ и сообщение системы о том, что модуль с заданным именем в исходном виде (чему соответствует SOUR) помещен в ЛБ. ССПМ информирует также о возможных нерегулярных ситуациях: модуль с заданным именем уже находится в ЛБ; в ЛБ нет свободного места для занесения модуля и др.

При переполнении ЛБ система автоматически ее уплотняет.

Занесение объектного вида модуля в ЛБ выполняется оператором

LOAD MOD ⟨имя ЛБ⟩. ⟨имя модуля⟩ OBJ.

Сообщения ССПМ и возможные нерегулярные ситуации аналогичны описанным выше.

Для того чтобы объектный модуль, ранее занесенный в личную библиотеку ССПМ, стал доступен системе для дальнейшей обработки (например, при сборке агрегата), его необходимо описать оператором

DSCR PLIB MOD ⟨имя ЛБ⟩. ⟨имя объекта⟩ OBJ.

Объявление модулей ЛБ. Если модули были записаны в ЛБ на предыдущих сеансах, то начало их использования в данном сеансе следует объявить оператором

DSCR PLIB MOD ⟨имя ЛБ⟩. ⟨имя модуля⟩ SOUR.

По этому оператору модуль становится доступным для обработки другими операторами ЯМК (трансляции, редактирования и т. д.).

Трансляция объектов. Трансляция исходного модуля выполняется оператором

TRANS [⟨имя ЛБ⟩.] ⟨имя объекта⟩.

Может быть указан целый список объектов. В качестве дополнительных параметров используются различные операции — вывод листинга, диагностических сообщений и т. д. По завершении трансляции система выдает запрос о выдаче и просмотре листинга на экран.

Корректировка объектов. Объекты, находящиеся в ЛБ, корректируются оператором

CORR PLIB ⟨имя ЛБ⟩ . ⟨имя модуля⟩.

Редактирование паспорта исходного текста модуля —

CORR PAS PLIB ⟨имя ЛБ⟩ . ⟨имя модуля⟩.

7.1.4. СОЗДАНИЕ ПРОГРАММНОГО АГРЕГАТА

Программный агрегат создается из модулей, находящихся в библиотеках ССПМ. Его структура представляется в виде графа, в вершинах которого указываются имена объектов (модулей, макромодулей, модулей данных), а на дугах — типы отношений, указывающие на управление структурой ПП (оверлейной, динамической, простой, подзадачный). Граф описывается оператором

$$\text{LINK}\left\{\begin{matrix}\text{PROG}\\ \text{SEG}\end{matrix}\right\}\ \langle\text{имя агрегата}\rangle\ (\langle\text{имя корневого модуля агрегата}\rangle)\#.$$

На его основе система генерирует внутреннее представление в виде матрицы смежности, а также в форме, удобной для представления на экране дисплея.

Сборка и отладка ПП. Сборка осуществляется с помощью оператора LINK, в котором указывается перечень модулей, входящих в собираемый агрегат. При этом модули, входящие в состав ПП (или отдельные компоненты), могут быть написаны на любом из ЯП ОС. Связи между ними указывают на образование агрегата различной структуры. Для тестирования ПП используется оператор TEST. Если корневому модулю ПП (или его части), подлежащему тестированию, не передаются никакие параметры, то для трассировки передач управления и данных создается отладочный вариант агрегата простой структуры

TEST PROG ⟨имя отладочного агрегата⟩ (⟨имя корневого модуля⟩)0.

Для организации тестирования ПП или его цепочек, корневые модули которых имеют параметры, в среде ССПМ разрабатывается ТЕСТ-модуль, моделирующий вызов и передачу параметров корневому модулю ПК. Он должен иметь паспорт, оформленный согласно требованиям ССПМ и содержащий в разделе вызываемых модулей имя корневого модуля ПК.

В этом случае отладочный вариант агрегата простой структуры создается оператором

TEST PROG ⟨имя отладочного агрегата⟩ (⟨имя ТЕСТ—модуля⟩)0.

В дальнейшем при выполнении собранного агрегата (ПП или его части) в среде ССПМ имена модулей, на которые передается управление, а также значения передаваемых им параметров будут выдаваться на экран терминала.

Оператором TEST могут быть созданы отладочные агрегаты простой, оверлейной, динамической и смешанной структур. Сборка ПП из отдельных исходных, объектных или загрузочных модулей с автоматической организацией межъязыкового интерфейса для разноязыковых модулей, но без включения средств трассировки выполняется оператором LINK.

Создание ПП простой структуры осуществляется оператором

LINK SEG ⟨имя корневого модуля⟩ (⟨имя корневого модуля⟩).

Полученный сегмент в загрузочном виде может быть помещен в ЛБ ССПМ для дальнейшего использования (например, при сборке ПП по частям).

Если необходимо выполнение собранного сегмента в среде ССПМ, то в случае, когда корневой модуль сегмента принимает (передает) параметры, необходимо, согласно требованиям системы, создать ТЕСТ-модуль, моделирующий вызов и передачу параметров корневому модулю сегмента. Оператор сборки в этом случае имеет вид

LINK SEG ⟨имя ТЕСТ-модуля⟩ (⟨имя ТЕСТ-модуля⟩).

Подготовка тестовых данных. Задание для вызова ССПМ должно содержать описание данных, которые могут использоваться программами, выполненными в среде системы в данном сеансе работы. Входные данные могут быть реально существующими и будут использованы программой при ее выполнении. В то же время нет необходимости, чтобы все входные данные существовали заранее и сразу при запуске были описаны.

Входные (тестовые) данные могут готовиться в соответствии с требованиями ЯП того модуля, в котором производится их ввод, и оформляются в виде модуля данных. Он снабжается паспортом и записывается следующим образом:

```
МОДУЛЬ DATA
ПАСПОРТ DATA
ЯЗЫК ⟨название ЯП⟩
ТЕКСТ
{данные в ЯП}
МКОН.
```

Оформленные таким образом данные помещаются в ЛБ оператором LOAD, корректируются оператором CORR и т. п. Перезапись данных из модуля в файл, используемый в программе в качестве входного, производится оператором ЯМК.

Тестирование ПП. Выполняется согласно плану сборки и тестирования на тестовых данных. Если агрегат (ПП или его

часть) в загрузочном виде находится в ЛБМ, то его тестирование осуществляется оператором

EXEC ⟨имя агрегата⟩.

Если агрегат содержится в ЛБ, то применяется оператор

EXEC ⟨имя ЛБ⟩ . ⟨имя агрегата⟩.

Для трассировки передач управления и значений (предусмотренных оператором TEST) используется оператор

EXEC # ⟨имя агрегата⟩

или

EXEC # ⟨имя ЛБ⟩ . ⟨имя агрегата⟩.

Результаты отображаются на экране терминала пользователя.

Просмотр результатов выполнения. Просмотр файлов за экраном выполняется с помощью оператора

LIST ⟨имя файла⟩.

Операнд ⟨имя файла⟩ является именем, описывающим просматриваемый файл. Оператор LIST рекомендуется применять при отладке для оперативного просмотра результатов счета. Просмотр файла за экраном терминала осуществляется при помощи команд просмотра листинга подобно тому, как это делается при просмотре листингов трансляторов в операторе TRANS.

Данное средство может быть также использовано для просмотра дампа аварийного завершения задач.

Распечатка результатов. Оператор печати файла предназначен для получения твердой копии файла на АЦПУ:

PRINT ⟨имя файла⟩.

Операнд ⟨имя файла⟩ аналогичен такому же в операторе LIST. Возможности использования оператора PRINT и требования к организации файла такие же, как и в операторе LIST.

В ССПМ имеется также возможность печати фрагментов листингов файлов после их просмотра за екраном дисплея. С этой целью набор операторов дополняется оператором «ПЕЧАТЬ».

Запись созданного ПП в ЛБ. Созданный ПП записывается в ЛБ с помощью оператора

LOAD MOD ⟨имя ЛБ⟩ . ⟨имя ПП⟩ LDR.

Запись же ПП в БЗМ ОС выполняется оператором SUBSYS. Работа организована в форме диалога с пользователем. По запросу пользователя выполняются подключение БЗМ ОС к ССПМ и перепись в нее агрегата, сформированного в ВБМ либо ЛБ, без уничтожения их содержимого. Выгруженный агрегат может в дальнейшем функционировать в среде ОС.

Завершение работы системы осуществляется по оператору SEND или CANCEL. Оператор SEND завершает сеанс работы с возможностью начала нового сеанса за тем же терминалом, а CANSEL отключает терминал от системы.

Результатом работы системы может быть протокол, отражающий последовательность действий при выполнении операций, связанных со сборкой, тестированием и отладкой. Полученные результаты сравниваются с данными из таблиц тестов. Вся информация о ходе тестирования и отладки фиксируется в журнале регистрации оши-

бок. Обнаруженные в ПП ошибки могут быть исправлены средствами системы (оператор CORR) в новом сеансе, а сам ПП — повторно тестирован.

7.2. ТЕХНОЛОГИЧЕСКИЙ МОДУЛЬ СППО

Рассматриваемый технологический модуль предназначен для выполнения:

автоматизированного построения функционального и системного наполнения методо-ориентированных ППП жесткой структуры;

автоматизированного получения программы решения задачи в проблемной области путем составления графа задачи;

решения задачи с помощью разработанного пакета.

Инструментальной основой ТМ является комплекс программных средств СППО, выполняющий автоматизированное построение программ решения задач на основе формализованных спецификаций модулей функционального наполнения ППП.

Структурно комплекс включает в себя две функциональные подсистемы: проектирования и управления процессом создания ППП, создания и ведения элементов функционального наполнения. В функции первой входят: формирование описания пакетов, проектируемых в среде комплекса; ввод текстов модулей функционального наполнения и данных, необходимых для решения задач с помощью пакетов; формирование описания информационных файлов и задач; постановка и решение задач в терминах предметной области; внесение изменений в исходные объекты функционального наполнения.

В функции второй подсистемы входят: ввод ПТД, описывающих элементы функционального наполнения; подготовка и ведение элементов в базе данных объектов; построение на основе ПТД спецификаций модулей и их отладка; сборка разноязыковых модулей и отладка агрегатов для задач ППП; отторжение агрегатов в загрузочном виде от среды подсистемы; информационно-справочное обслуживание, предоставляющее пользователю сведения об элементах функционального наполнения.

В качестве этой компоненты может использоваться ТМ ССПМ, который представляет собой, по существу, первый уровень автоматизации построения ППП. Создаваемый на его основе программный продукт — это набор программ решения задач ППП, разработанных по модульному принципу и содержащих модули на ЯП в базе данных объектов (банк модулей или личные библиотеки пользователей).

СППО предоставляет пользователю возможность получить ППП жесткой структуры. На входном языке пакета пользователь выполняет постановку задач и решает их.

Функцию управления работой СППО выполняет монитор, принимающий решение о подключении каждой из подсистем для обработки запросов пользователя. Работа осуществляется в диалоговом режиме. Технологическую основу рассматриваемого ТМ составляют методы и средства, поддерживающие стратегию синтезирующего программирования с соблюдением принципа модульности.

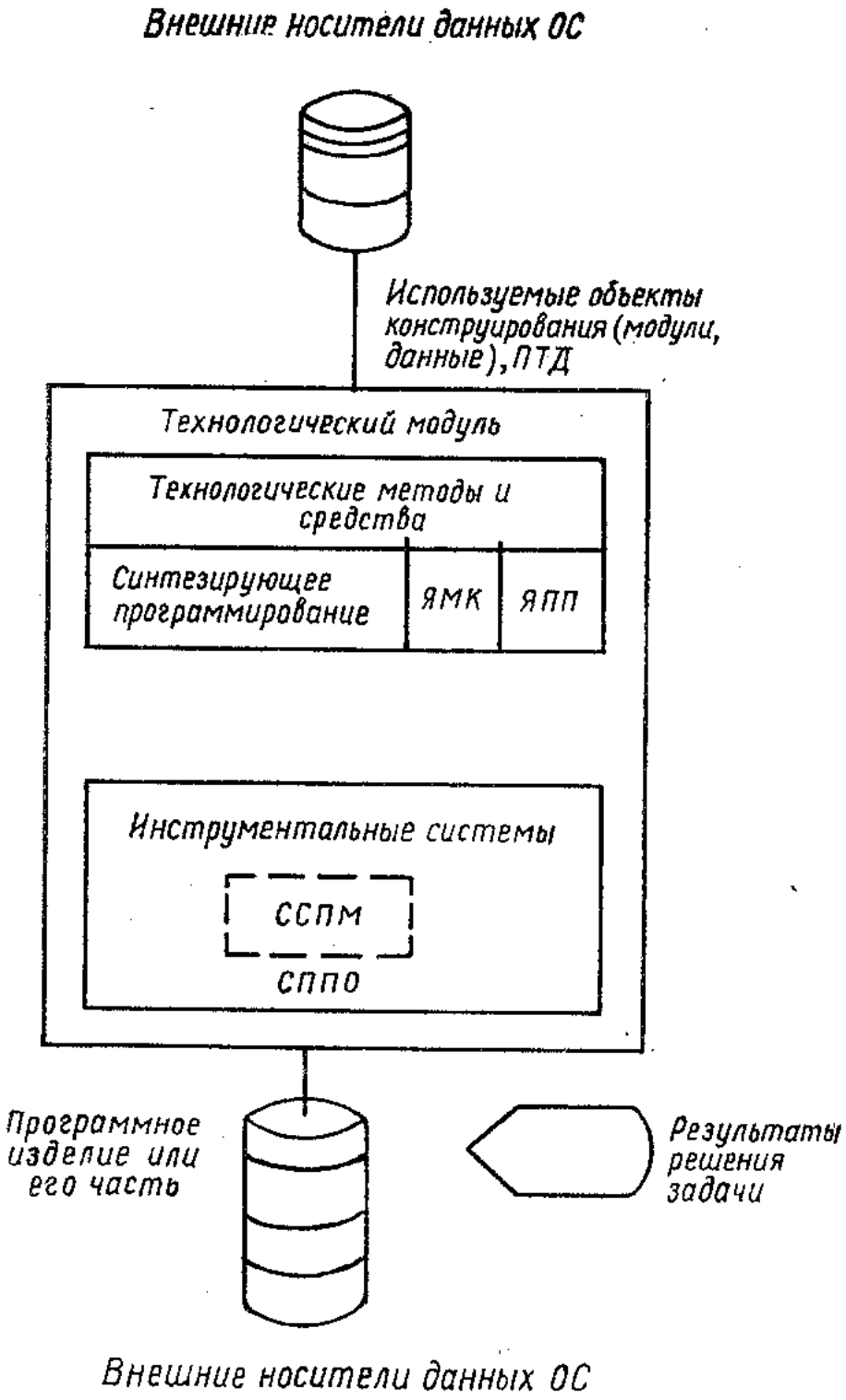


Рис. 7.2. Состав ТМ разработки методо-ориентированных ППП

В качестве базовых технологических средств используются языки: модульного конструирования программ, предоставляемых ССПМ; проектирования пакетов, предоставляемых подсистемой проектирования и управления.

В дальнейшем описываются только те возможности данного ТМ, которые обеспечивают выполнение функций проектирования и управления созданием ППП. Схематически состав данного ТМ представлен на рис. 7.2.

7.2.1. ПРИМЕНЕНИЕ ТМ ДЛЯ РАЗРАБОТКИ ППП

Применение ТМ состоит в выполнении следующих операций (полный набор приведен в табл. 7.2): подготовка исходной информации; ввод исходных модулей и модулей данных в среду комплекса; составление описаний задач и файлов данных для задач; решение задач; завершение работы комплекса. Кроме этих могут выполняться сервисные операции, поддерживаемые комплексом.

Таблица 7.2

Номер операции	Операция	Оператор языка комплекса
1	Подготовка исходной информации	—
1.1	Подготовка исходных наборов данных, содержащих тексты используемых модулей и данные для решения задач	—
1.2	Подготовка комплекса к запуску	—
1.3	Запуск комплекса	SUPER
1.4	Начало работы	ПАКЕТ
2	Ввод модулей в среду комплекса	ВМ или ВМД
3	Ввод данных в среду комплекса	ВД или ВДД
4	Составление описаний задач ППП	ЗАДАЧА
5	Составление описаний файлов данных для задач	ФАЙЛ
6	Решение задач	СЧЕТ, ВЫВД ВЫВП
7	Сервисные технологические операции	СТЕРЕТЬ, ВМД ВДД
8	Завершение работы комплекса ССПО	КП

Подготовка исходных наборов данных. Исходной информацией для СППО являются программные модули и их спецификации (паспорта), сформированные по требованиям ССПМ в виде порций ввода. Порции ввода располагаются во внешних файлах.

В случае, если предварительная разработка отдельных элементов функционального наполнения выполняется средствами ССПМ с использованием ЛБ, исходные тексты модулей и их спецификации должны переписываться в наборы данных ОС.

Подготовка задания на запуск ССПО. Запуск комплекса должен включать описание: данных с исходными текстами модулей задач и модулей данных; терминалов пользователей комплекса в данном сеансе работы; данных для задач, решаемых средствами ППП.

Запуск СППО осуществляется средствами ОС. В случае, если запуск произведен успешно, СППО вступает в диалог с пользователем по уточнению режима работы.

После завершения настройки системы на заданный режим работы вводится оператор SUPER (фактическое начало сеанса работы), что означает начало (либо продолжение) конструирования ППП.

Для этого используется оператор ПАКЕТ в одном из следующих форматов:

ПАКЕТ ⟨имя пакета⟩ НАЧАТЬ
либо
ПАКЕТ ⟨имя пакета⟩ ПРОДОЛЖИТЬ.

Для работы с СППО в режиме решения задач изготовленным пакетом используется оператор ПАКЕТ в формате

ПАКЕТ ⟨имя пакета⟩ ВЫПОЛНИТЬ.

В в о д м о д у л е й в с р е д у С П П О. Исходные тексты и спецификации модулей вводятся в среду СППО с помощью оператора языка проектирования пакетов. Для ввода модулей из файлов ОС используется оператор

ВМ [⟨имя файла⟩].

В нем указывается имя одного из файлов с исходной информацией. Оператор следует вводить столько раз, сколько файлов будет обрабатываться в данном сеансе.

Для ввода текстов и паспортов модулей непосредственно с экрана дисплея используется оператор

ВМD [⟨имя модуля⟩] П[АСПОРТ]/ Т[ЕКСТ].

По умолчанию предполагается, что вводится текст модуля. Допускается раздельный ввод текстов модулей и паспортов.

После ввода исходной информации СППО автоматически выполняет трансляцию модулей и сохранение результатов трансляции в хранилищах системы.

В в о д д а н н ы х в с р е д у С П П О. Входные данные (модули данных), как и исходные модули, оформляются в виде порций ввода в файлах ОС и вводятся в систему оператором

BD [⟨имя файла⟩].

Введенные модули данных транзитом пересылаются в библиотеки данных СППО. В случае ввода данных непосредственно с экрана дисплея используется оператор

BDD [⟨имя модуля данных⟩].

Имена модулей данных, а также имена функциональных модулей могут иметь синонимы, соответствующие понятиям предметной области разрабатываемого пакета. Синонимы задаются вслед за именем модуля в круглых скобках:

ТЕКСТ SKVIT (ВЕКТОР-ПЛОЩ).

О п и с а н и е з а д а ч. Все задачи, на решение которых ориентируется разрабатываемый ППП, должны быть описаны с помощью оператора

ЗАДАЧА ⟨имя задачи⟩ [ВХ ␣(список исходных данных)]
[⟨описание процесса сборки⟩].

В нем указываются имена файлов исходных данных, которые будут использоваться при решении задачи, и описывается процесс сборки функционального наполнения пакета.

При описании процесса сборки указываются имя агрегата, отождествляемого с решаемой задачей, и список элементов функционального наполнения, составляющих агрегат. Первым в списке указывается корневой модуль. Порядок перечисления остальных элементов (модулей, программ) в списке не существен.

Связь задачи с исходными данными, оформленными как модули, осуществляется косвенно, т. е. с использованием логических файлов СППО, указанных в операторе ЗАДАЧА. Логический файл описывается оператором ФАЙЛ. Связь задачи с модулем данных осуществляется на этапе решения (счета) задачи. При этом информация из модуля данных автоматически переносится в набор данных, указанный в операторе ФАЙЛ.

Описание файлов данных. Для описания логических файлов используется оператор

ФАЙЛ ⟨имя файла⟩ ⟨имя модуля данных⟩.

С его помощью описываются как входные, так и выходные логические файлы. Оператор ФАЙЛ для входных файлов выполняется столько раз, сколько модулей данных используется для наполнения одного функционального набора данных (и так для каждого функционального набора данных).

Решение задач. Выполняется посредством оператора

СЧЕТ ⟨имя задачи⟩ [⟨список исходных данных задачи⟩].

Каждому оператору СЧЕТ соответствует оператор создания задачи ЗАДАЧА. Если на шаге счета необходимо изменить некоторые исходные данные (т. е. входные логические файлы), указанные в операторе ЗАДАЧА, то они перечисляются в последовательности, установленной оператором ЗАДАЧА, а данные, не требующие изменения, отделяются запятой.

Результаты счета могут просматриваться за экраном терминала пользователя с помощью оператора

ВЫВ D {⟨список имен файлов⟩}.

Вывод результатов на печать осуществляется оператором

ВЫВП {⟨список имен файлов⟩}.

Оператор ВЫВП используется в том случае, если выходной файл результатов для вывода на печать не является стандартным, а представляет собой набор данных на МД или МЛ. В противном случае он выводится на АЦПУ автоматически.

Сервисные ТО. Ими являются операции манипулирования элементами разрабатываемого пакета:

корректировка функциональных модулей (оператор BMD);

корректировка модулей данных (оператор BDD);

уничтожение модуля (оператор CTM) и данных (оператор CTD);

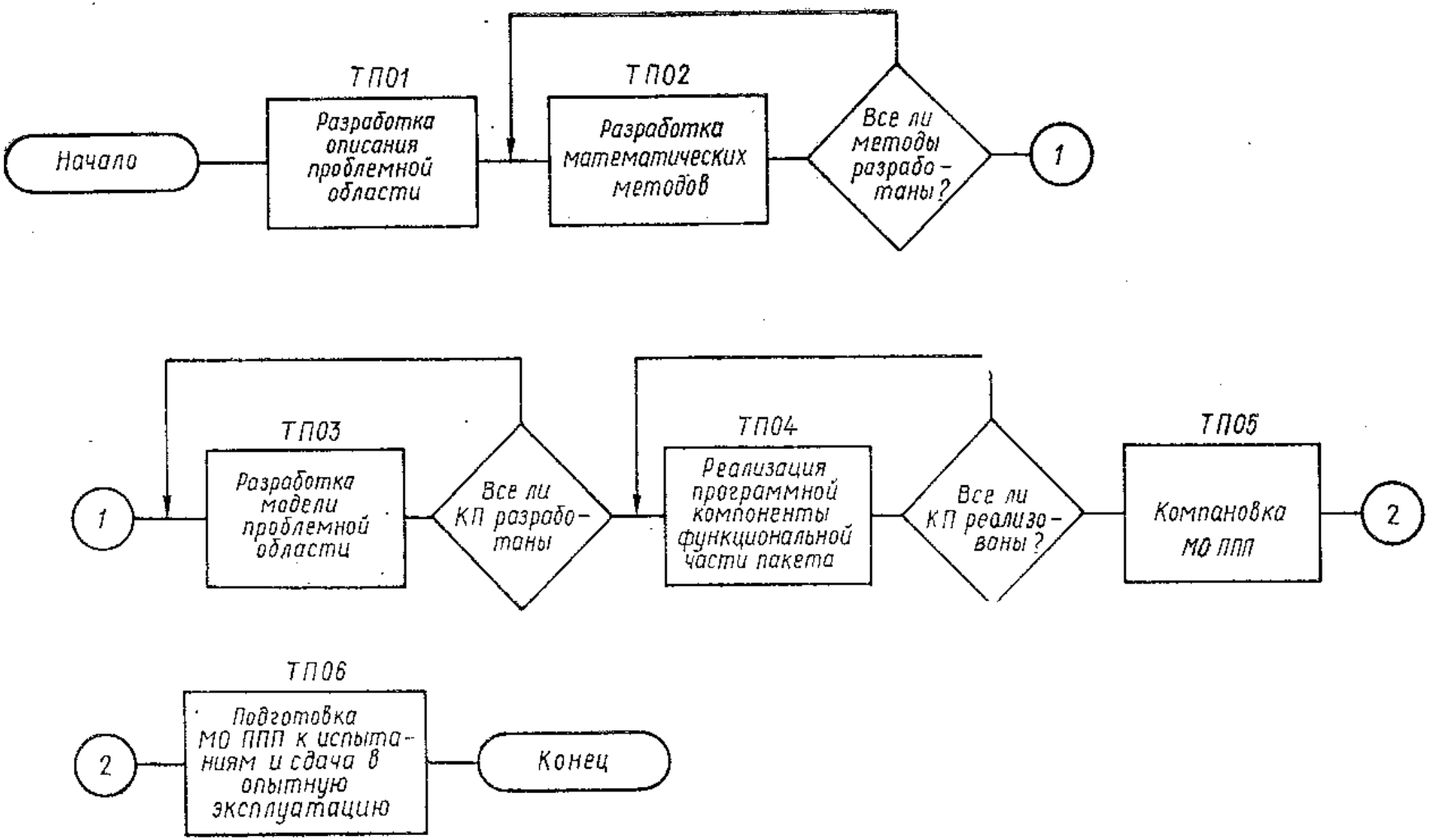


Рис. 7.3. Технологический маршрут и карта технологической линии

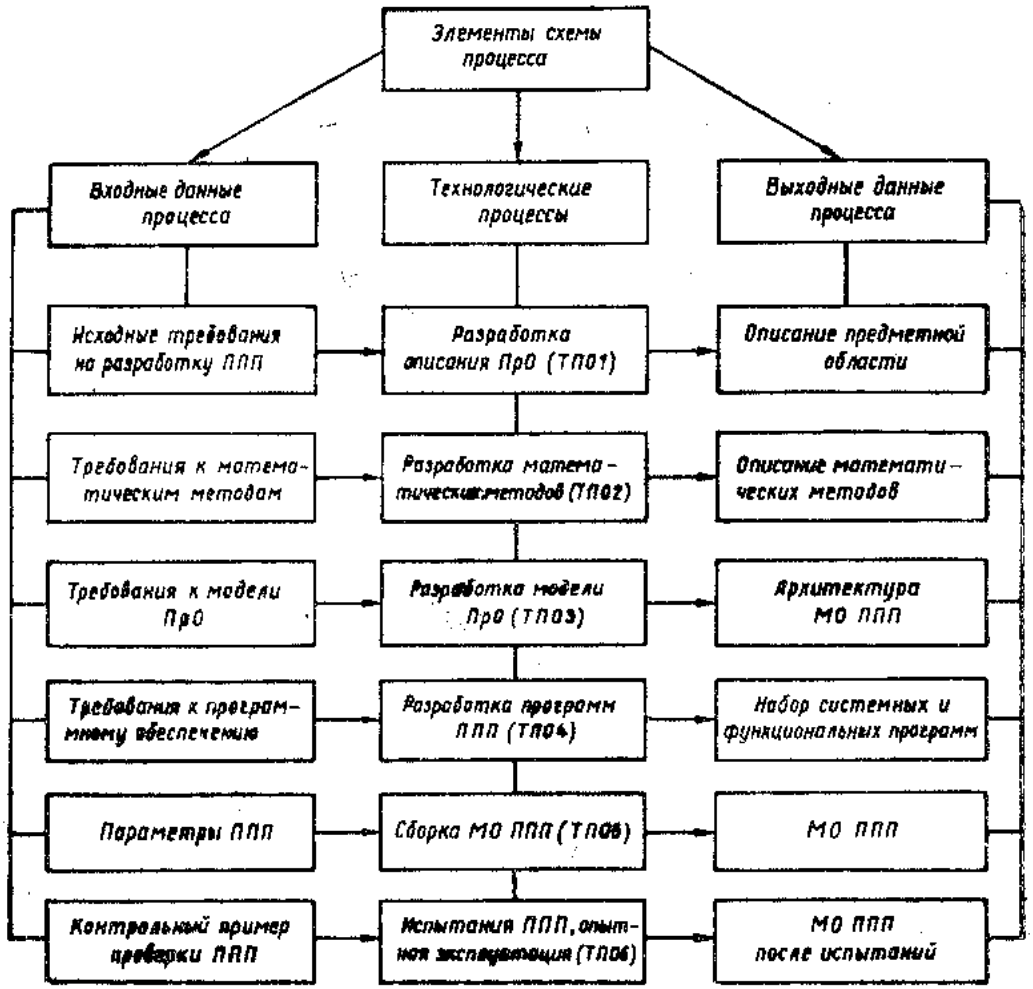


Рис. 7.4. Схема процесса разработки методо-ориентированных ППП

уничтожение логических файлов (оператор СТФ), задач (оператор СТЗ) и пакета (оператор СТП).

Перечисленные операторы используются также для исправления ошибок, допущенных и выявленных в процессе создания пакета.

З а в е р ш е н и е р а б о т ы С С П О. Осуществляется оператором КП[АКЕТ] без операндов. При его получении монитор передает управление ССПМ и допускает ввод операторов ЯМК. Возврат в среду проектирования пакета осуществляется при вводе оператора ПАКЕТ. Полное завершение работы ССПО выполняется оператором SEND или CANCEL ССПМ.

7.3. ТЕХНОЛОГИЧЕСКИЕ ПРОЦЕССЫ РАЗРАБОТКИ ППП

Выше описаны состав и функции ТМ разработки ППП. Для полноты изложения необходимо привести описания отдельных технологических процессов, входящих в конкретную ТЛ. В качестве такой ТЛ выберем проектирование методо-ориентированных ППП (технологический маршрут и карта приведены на рис. 7.3 и 7.4) и на ее примере определим основные ТП.

7.3.1. ТП «РАЗРАБОТКА ОПИСАНИЯ ПРОБЛЕМНОЙ ОБЛАСТИ» (ТП01)

Общая характеристика. Исходными данными для выполнения данного ТП является информация, содержащаяся в документе «Исходные требования на разработку ППП». Информация включает описания постановок задач ПрО, основанные на знаниях о ПрО, которыми обладают проблемные программисты и постановщики задач. Целью выполнения ТП01 является:

определение множества постановок задач ПрО;

формализация задач, т. е. выбор и разработка тематических моделей, адекватных задачам;

установление соответствий между понятиями, фигурирующими в постановках задач, их характеристиками и параметрами математической модели;

определение множества входных и выходных понятий ПрО.

В завершение процесса составляется документ «Описание проблемной области», содержащий описание постановок задач ПрО. Описание — это спецификация задачи, которая оформляется на дополнительном бланке ПТД. Оно содержит следующую информацию: имя задачи; описание функции и математической модели задачи; описание понятий ПрО, включая их имена, статус и семантику. Информация о спецификациях задач накапливается в процессе выполнения операций данного ТП.

Организационная структура. Приведена на рис. 7.5 и включает состав исполнителей, связи между которыми определяют действия при выполнении работ на процессе. Руководитель группы разработчиков уточняет исходные требования на создание МО ППП, затем совместно с математиком-аналитиком определяет множество задач, входных и выходных понятий. Математики разрабатывают математические модели задач.

Работы, выполняемые математиками, контролируются математиком-аналитиком. Контроль выполнения данного процесса проводится с целью проверки полноты и непротиворечивости описания ПрО.

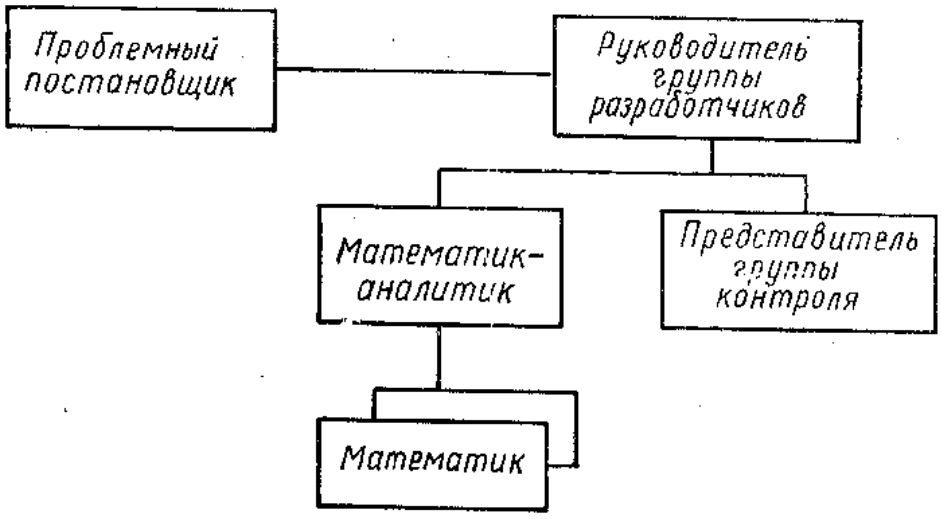


Рис. 7.5. Организационная структура ТП01

Состав ТП01. Процесс состоит из четырех операций, описываемых ниже.

Операция ТО01 предназначена для анализа исходных требований на разработку МО ППП. В результате ее выполнения формируется информация об именах задач ПрО и их функциях, которая фиксируется на бланке ПТД.

Операция ТО02 предназначена для выделения множества понятий для каждой задачи и присвоения понятиям имен, их возможных статусов (типа входной или выходной). Операция обладает цикличностью и может выполняться группой математиков для каждой задачи последовательно либо параллельно. В результате ее выполнения определяется вид информации, вносимой в выходной документ, связанный с описанием понятий.

Операция ТО03 — это последовательность действий по подбору и разработке математических моделей, адекватных задачам ПрО. При этом устанавливается соответствие между понятиями, определенными в постановке задачи, и параметрами математической модели. При выполнении операции не исключается дополнение или уточнение проведенных описаний понятий. Операция обладает цикличностью и может выполняться группой математиков для каждой задачи последовательно либо параллельно. Результатом выполнения является информация, содержащая описание математических моделей, которая заносится в выходной документ.

Операция ТО04 предусматривает проверку требований, предъявляемых к разрабатываемым МО ППП:

полноту множества задач, решенных в данной ПрО;

адекватность математических модулей задач;

непротиворечивость параметров математических моделей понятиям, фигурирующим в постановках задач.

Результаты проверки фиксируются в протоколе контроля и используются при повторном выполнении ранее рассмотренных операций в соответствии с инструкцией.

7.3.2. ТП «РАЗРАБОТКА МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ» (ТП02)

Общая характеристика. Целью данного процесса являются выбор существующих и разработка новых методов решения задач ПрО. На основании анализа математических моделей задач и исходных требований к ним выбираются современные методы решения, наиболее полно удовлетворяющие исходным требованиям. Если их не удается найти, то предусматривается операция разработки новых методов решения задач. Данный процесс выполняется математиком-аналитиком с привлечением математиков. Результатом является документ «Описание математических методов», содержащий полное описание методов и условий их применения для каждой задачи ПрО.

Организация структуры. Выполняется составом исполнителей, приведенным на рис. 7.6. Операции ТП02 выполняются математиком-аналитиком с привлечением математиков. Операция

контроля выполняется представителем группы контроля при участии математика-аналитика.

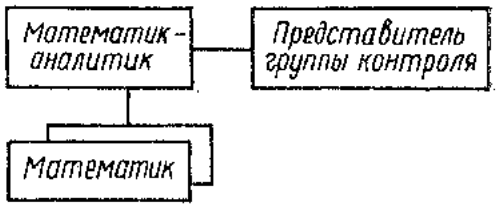


Рис. 7.6. Организационная структура технологического процесса ТП02

Состав ТП02. Данный процесс состоит из четырех операций, семантика которых приводится ниже.

Операция ТО01 заключается в проведении глубокого анализа современных методов решения задач в данной ПрО, наиболее полно удовлетворяющих их требованиям. Если такие методы найдены, то проводится обоснование выбора, заключающееся в характеристике метода и описании условий его применимости. В противном случае формулируется обоснование невозможности его применения, оформляется задание на разработку новых методов или доработку существующих с целью удовлетворения всем поставленным требованиям.

В результате выполнения данной операции составляется полное описание выбранных методов со ссылками на источники, а также описание условий их применения для каждой задачи.

Целью **операции ТО02** является выполнение заданий на разработку новых методов решения задач. Они составляются при выполнении предыдущей операции в тех случаях, если ни один из существующих современных методов не может быть применен для решения какой-либо задачи ПрО. Для вновь разрабатываемых методов проводится теоретическая разработка с полным описанием метода и условий его применения, а также строгое математическое доказательство корректности и адекватности в условиях, определяемых спецификой ПрО.

Если новые методы, полностью удовлетворяющие поставленным требованиям, разработать не удается, то по согласованию с математиком-аналитиком и руководителем разработки принимаются методы, наиболее полно удовлетворяющие требованиям.

Операция ТО03 выполняется с целью проверки непротиворечивости выбранных и разработанных методов поставленным требованиям. Выполняет эту операцию представитель группы контроля, имеющий соответствующую квалификацию, при участии математика-аналитика.

Операция ТО04 является завершающей в данном процессе и выполняется с целью оформления документа «Описание математических методов» на основе рабочих документов, разработанных при выполнении операций ТОθ1 и ТОθ2.

7.3.3. ТП «РАЗРАБОТКА МОДЕЛИ ПРОБЛЕМНОЙ ОБЛАСТИ» (ТП03)

Общая характеристика. Целью данного процесса является создание функциональной и системной архитектуры ППП. Для разработки функциональной архитектуры ПрО проводится разбиение каждого из выбранных методов решения задач на отдельные компоненты, реализующие конкретные функции, называемые функциональной спецификацией. Она отражает иерархию функций комплекса

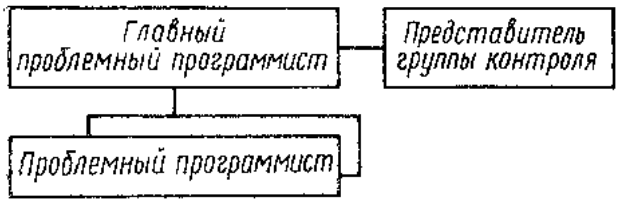


Рис. 7.7. Организационная структура технологического процесса ТП03

программ (КП) реализации метода и содержит формализованное описание функций компонент, их информационные и управляющие связи.

Разработка системной архитектуры состоит в создании графовой структуры КП и оформлении спецификаций на комплексы программ и модули, входящие в них. Графовая структура включает модули и типы взаимосвязей между ними. Спецификации на КП и входящие в него модули содержат формализованное описание: функционального назначения модуля; входных и выходных данных; связей с другими модулями и т. п.

Для представления спецификаций используется язык проектирования программ. Спецификации оформляются в виде комплекта ПТД. При проектировании системной архитектуры КП используется принцип многоуровневой нисходящей методологии проектирования сложных программ. Процесс завершается подготовкой выходного документа «Системная архитектура модели ПрО», содержащего структурные схемы КП и спецификации на все модули, и КП в целом.

Организационная структура. Приведена на рис. 7.7, обеспечивает проектирование системной архитектуры модели ПрО и подготовку выходных документов по процессу. Основную функцию выполняют проблемные программисты под руководством главного проблемного программиста. Представитель группы контроля при участии главного проблемного программиста проводит контроль соответствия системной архитектуры ПрО функциональной, который состоит в проверке сохранения функциональной направленности в графовой структуре и непротиворечивости использования данных модулями, входящими в структуру.

Состав ТП03. Процесс состоит из четырех операций, описываемых ниже.

Операция ТО01 выполняется с целью разработки функциональных спецификаций на КП. Суть ее состоит в проведении детального анализа задач и методов их решения с точки зрения их программной реализации в конкретной вычислительной среде. Анализ методов решения и детализация их на автономные компоненты обеспечивают реализацию конкретного метода. В результате формируется представление о модели КП как о некоторой его структуре. Функциональные спецификации являются результатом операции и должны содержать:

иерархию функций (компонент) КП и их описание;

описание связей по управлению и информации между компонентами, включая входные и выходные данные, их типы и формы представления в вычислительной среде;

описание требований и ограничений к реализации КП.

Функциональные спецификации представляются в виде архитектур с-функциональных схем, выполненных с использованием диа-

грамм Лейтона, HIPO-диаграмм, SA-диаграмм и т. п. Операция выполняется применительно ко всем методам, перечисленным в документе «Описание проблемной области».

Операция ТО05 состоит из анализа и детализации компонент КП с целью выделения программных модулей и определения их взаимосвязей. Выделяемые модули должны удовлетворять следующим требованиям:

выполнять одну достаточно простую функцию или несколько функций, работающих с одной и той же структурой данных;

иметь один вход, один выход;

возврат управления осуществлять в точку вызова вызвавшего модуля;

передавать данные через механизмы вызова типа CALL (стандартный тип вызова);

размер модуля должен быть ограничен страницей текста.

Выполнение данных требований упрощает дисциплину взаимодействия программистов, закладывает основы распараллеливания работ по программированию и документированию программ. Результатом выполнения такой операции является часть системной архитектуры модели ПрО, а именно графовая модель КП, разрабатывающаяся согласно методологии СППО, и ПТД. Эти документы представляют собой паспорт КП и модулей и содержат их общее описание.

Операция ТО10 предназначена для заполнения ПТД, которые оформляются в КП и отдельные модули, включая описание логики выполнения модулей. Созданные ПТД включаются в состав системной архитектуры модели ПрО.

Системная архитектура, разработанная на основе данной операции, уточняется и дополняется при выполнении следующих процессов. В частности, могут доопределяться некоторые разделы спецификаций и вводиться новые документы, сопровождающие этапы программирования, сборки и тестирования КП.

Операция ТО15 предназначена для экспертного контроля хода разработки системной архитектуры модели ПрО, заключающегося в проверке соответствия системной архитектуры функциональной архитектуре модели ПрО. Контроль выполняется представителем группы контроля с участием главного проблемного программиста.

7.3.4. ТП «РЕАЛИЗАЦИЯ ФУНКЦИОНАЛЬНОЙ ЧАСТИ ПАКЕТА» (ТП04)

Общая характеристика. В данном процессе выполняются работы по программированию, отладке и тестированию КП. Согласно рассматриваемой технологии, эти работы выполняются в два этапа.

Первый этап — реализация отдельных модулей. Для распараллеливания работ по программированию, отладке и тестированию модулей главный проблемный программист привлекает группу проблемных программистов.

На втором этапе производится сборка отлаженных модулей в комплекс в соответствии с его графовой структурой.Затем производится

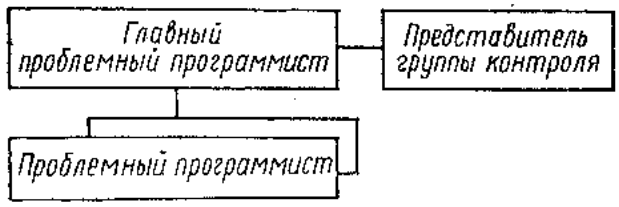


Рис. 7.8. Организационная структура технологического процесса ТП04

тестирование интерфейса модулей. Качество выполнения работ зависит от того, насколько тщательно на предыдущем процессе были разработаны граф КП и спецификации на модули и КП. Процесс считается выполненным в том случае, если проведено тестирование всех КП, реализующих задачи ПрО.

Организационная структура. Приведена на рис. 7.8. В ней этапы программирования, отладки и тестирования модулей выполняются проблемными программистами. Контроль за выполнением этих работ, сборку и тестирование КП выполняет главный проблемный программист с привлечением проблемных программистов, а также представитель группы контроля при участии главного проблемного программиста. Контроль заключается в установлении соответствия спецификации и графа КП результатам тестирования.

Состав ТП04. Процесс состоит из трех операций, описываемых ниже.

В **операции ТО01** объединены работы по программированию, отладке и тестированию модулей (компонент) КП. Программирование выполняется на ЯП, способствующем получению наиболее эффективной программы. Производительность труда программиста при этом предполагается высокой. Последний показатель зависит от того, насколько тщательно была разработана системная архитектура при выполнении процесса ТП03. Реализация программных модулей производится по принципу структурного проектирования, при котором программа представляется в виде графовой структуры и независимо от формы представления каждый из ее операторов заменяется конструкциями ЯП.

Отладка программ начинается после исправления синтаксических ошибок и ошибок программирования, выявленных при тестировании программ. При этом используются средства отладки, имеющиеся в арсенале средств ОС. Тестирование проводится для выявления семантических ошибок и ошибок проектирования КП и выполняется на основании тестов, оформленных согласно применяемой методике. Критерием завершенности тестирования модуля является корректная обработка всех содержащихся в тесте ситуаций.

Контроль выполнения работ по данной операции проводит главный программист при участии проблемного программиста. При этом проверяются соответствие программы спецификациям модуля и полнота тестирования.

Операция ТО10 предназначена для сборки и тестирования КП. Этап сборки и тестирования основан на общепринятой стратегии нисходящего тестирования с использованием драйверов и заглушек для моделирования взаимодействия модулей. Тестирование КП (или его частей) выполняется с использованием технологических методов и средств инструментальной системы ССПМ. Для его проведения со-

ставляются план и таблица тестов. План предусматривает тестирование на тестовых данных, моделирующих реальные ситуации, функций КП, связей по управлению и по данным.

Операция ТО15 предназначена для проверки соответствия текста КП его спецификациям и графовой структуре. Эту операцию выполняет представитель группы контроля при участии главного проблемного программиста.

7.3.5. ТП «КОМПОНОВКА МО ППП» (ТП05)

О б щ а я х а р а к т е р и с т и к а. Поскольку входной язык и системное наполнение пакета обеспечиваются ССПМ, то целью данного процесса является формирование модели функционального наполнения МО ППП. Операции автоматизированы и выполняются с использованием средств инструментального комплекса. Модель функционального наполнения — это множество исходных и модулей данных, представленных в библиотеках системы, а также каталоги модулей и задач.

Форма представления исходных модулей соответствует требованиям СППО. Результатом выполнения данного процесса является готовый к эксплуатации МО ППП, общение с которым осуществляется на входном языке пакета, являющемся подмножеством языка проектирования пакетов.

Рассматриваемый процесс выполняется главным проблемным программистом (рис. 7.9) с привлечением других проблемных программистов. Контроль хода выполнения процесса осуществляет представитель группы контроля при участии главного проблемного программиста. При этом проверяется соответствие описаний задач спроектированному графу КП.

С о с т а в ТП05. Процесс состоит из трех операций, которые содержательно описываются ниже.

Операция ТО01 является первой в реализации модели функционального наполнения МО ППП и выполняется с целью организации библиотеки исходных модулей и библиотеки модулей данных. Выполнение поддерживается средствами инструментального комплекса. Форма представления вводимых модулей соответствует требованиям СППО.

Операция ТО05 предназначена для создания каталогов задач и модулей. Для этого формируются и подаются на вход инструментального комплекса описания задач, которые составляются согласно требованиям инструментального комплекса с использованием соответствующих средств. Данная операция является завершающей операцией процесса формирования модели функционального наполнения МО ППП.

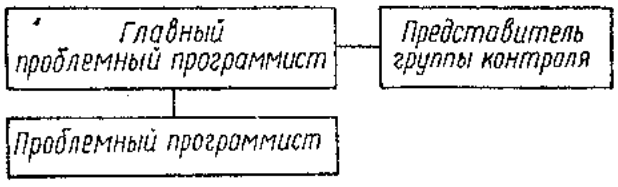


Рис. 7.9. Организационная структура технологического процесса ТП05

Операция ТО10 выполняется с целью подготовки контрольных примеров для апробации

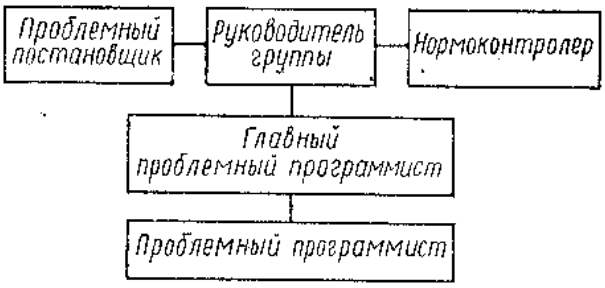


Рис. 7.10. Организационная структура технологического процесса ТП06

разработанного пакета. В качестве контрольных выбираются тестовые примеры, использовавшиеся для тестирования КП в ТП04. При работе с МО ППП используются средства языка проектирования пакетов СППО. Апробация пакета заключается в проведении счета задач и получении точных результатов.

Соответствие описаний задач сформированной модели функционального наполнения графу и исходным требованиям на разработку МО ППП проверяется (операция ТО15) представителем группы контроля при участии главного проблемного программиста.

7.3.6. ТП ПОДГОТОВКА МО ППП К ИСПЫТАНИЯМ (ТП06)

Общая характеристика. Целью выполнения процесса является проведение комплексных испытаний разработанного пакета на соответствие исходным требованиям и сдача его в опытную эксплуатацию. В результате составляется акт проведения испытаний МО ППП (опытного образца) и оформляется соответствующая эксплуатационная документация.

Организационная структура. Приведена на рис. 7.10. Главный проблемный программист совместно с руководителем разработки составляет программу и методику испытаний МО ППП. Они же готовят эксплуатационную документацию на пакет, привлекая, по мере необходимости, проблемных программистов, разрабатывавших документацию в стадии рабочего проектирования на отдельные части МО ППП.

После проверки подготовленных документов представителем нормоконтроля проводятся комплексные испытания пакета. В них участвуют руководитель группы разработки как отчитывающаяся сторона и проблемный постановщик как сторона, принимающая и оценивающая результат разработки.

Состав ТП06. Процесс состоит из четырех операций, содержательный смысл которых приводится ниже.

Операция ТО01 выполняется с целью составления программы и методики испытаний разработанного МО ППП. Последняя разрабатывается в соответствии с требованиями, предъявляемыми к методике испытаний МО ППП. При составлении программы испытаний используется описание тестовых примеров, разработанных для отладки КП функциональной части пакета. При этом выбираются примеры, наиболее полно охватывающие проверку КП и использующиеся в качестве контрольных примеров. Общий перечень их составляется таким образом, чтобы охватить все КП разработанного МО ППП.

Результатом выполнения операции является документ «Программа и методика испытаний».

Операция ТО05 выполняется с целью оформления всех документов рабочего проекта согласно требованиям стандартов ЕСПД в системе технической документации на АСУ. Разработка документов ведется на основании ранее подготовленных комплектов документации на стадии рабочего проекта функционального наполнения МО ППП и описаний системной архитектуры модели ПО.

Операция ТО10 выполняется для проведения нормоконтроля разработанных документов рабочего проекта. В результате будет получен готовый комплект документов рабочего проекта на МО ППП.

Операция ТО15 завершает ТП06 по сдаче в опытную эксплуатацию разработанного МО ППП. После проведения комплексных испытаний в соответствии с разработанной программой и методикой, а также подписания акта о проведении испытаний начинается опытная эксплуатация разработанного МО ППП.

Глава 8

ВОПРОСЫ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПС НА ОСНОВЕ ТЕХНОЛОГИИ СБОРОЧНОГО ПРОГРАММИРОВАНИЯ

В данной главе рассматриваются различные аспекты применения технологии сборочного программирования в разработке ПС с использованием понятия технологической линии. Рассматриваемые вопросы затрагивают организацию процесса разработки ПС, планирование, оценку качества создаваемого продукта и др.

Организационный аспект управления разработкой. В разработке программных систем принимают участие разные специалисты, связи между которыми в значительной степени влияют на результат разработки. В литературе предлагался и обсуждался ряд форм организации коллектива разработчиков [19, 149 и др.] — бригада главного программиста, хирургическая бригада, бригада без специализации и др. Однако практически в каждом коллективе складываются свои организационные формы, которые, как правило, определяются видом объекта и квалификацией руководителя разработки.

При этом основная задача организации разработки состоит в расстановке специалистов по видам программистских работ в соответствии с их квалификацией для получения качественного продукта и в срок. При расстановке необходимо иметь дополнительный ресурс (в случае болезни, увольнения специалистов и т. п.), способный продолжить разработку. Простое добавление специалистов в «пиковые» моменты разработки не только не приводит к желаемым результатам [19], но и снижает производительность тех, кто работает.

Поэтому в настоящее время при переходе к новым условиям хозяйствования необходимо исследовать не только бригадные формы работ, успешно себя проявившие в других областях народного хозяйства, но и факторы, влияющие на эффективность управления разработкой. К ним относятся:

виды и свойства разрабатываемого объекта;

набор процессов в ТЛ и их обеспеченность нормативно-методическими, регламентирующими и справочными материалами;

принципы планирования трудозатрат, учета и регулирования хода разработки;

организация и методы контроля работ;

нормирование труда, времени разработки и др.

Приведенные факторы характеризуют процессы разработки ПС основанные на применении любых технологий программирования В этой главе рассматриваются вопросы, связанные с применением метода сборочного программирования.

Планирование. Для соблюдения сроков разработки объекта во многих проектах проводится планирование разработки. Известно около 20 различных моделей планирования, включающих расчет трудозатрат, сроков и требуемого числа специалистов. Каждая модель разрабатывалась индивидуальным способом на основе использования накопленных данных о проведенных разработках. В качестве исходных данных и критериев в моделях применялись разные характеристики продукта и среды разработки. Большинство методов основано на прогнозировании объема продукта, выражаемого в числе строк (операторов, команд).

Предполагаемый объем делился на среднестатистическое значение производительности (число строк) одного программиста в год. В результате получалась планируемая трудоемкость. Производительность разработчиков ПС колеблется в больших диапазонах в зависимости от применения языков программирования высокого уровня (производительность повышается в 4—5 раз по сравнению с языком типа Ассемблер), форм организации работ в коллективе, режима работы программистов на ЭВМ, производительность которых повышается на 20 % при диалоговом режиме и возможности доступа к ЭВМ в любое время.

Эти данные получены в [176] при проведении исследований 12 моделей затрат на одном гипотетическом проекте и состоят в следующем:

отношения между самым низким и самым высоким значениями затрат выражаются как 1 : 7,6;

отношения между самым коротким и самым длинным сроком разработки проекта выражаются как 1 : 1,9.

Все модели проверялись с одинаковыми данными для производительности, сложности и др. в разных коллективах. Для определения числа разработчиков планируемая трудоемкость делилась на время разработки, которое определялось в зависимости от заданных сроков разработки объекта. Естественно, что такая модель расчета является очень поверхностной и, кроме того, зависит от организации управления разработкой и требованиями создания высокого качества программного продукта. Данное обстоятельство приводит к увеличению сроков разработки и к снижению производительности труда, что в конечном итоге увеличивает общие трудозатраты.

Контроль управления качеством программного продукта. Одним из основных требований к программным системам является обеспечение их высокого качества, что может быть достигнуто организацией системы управления качеством.

Под качеством программных продуктов будем понимать совокупность свойств, обуславливающих его пригодность к использованию по назначению при регламентации совокупности условий функционирования. Каждое свойство является его качественной характеристикой и в процессе разработки может выражаться определенными при-

знаками, которые могут оцениваться экспертным и аналитическим путями.

Для разных классов программных объектов на этапе разработки требований вырабатывается номенклатура показателей качества и устанавливаются их базовые значения в соответствии с имеющимся образцом. Показатели включаются в описание процессов, и на их основе проводится контроль качества.

Обеспечение качества представляет собой совокупность планируемых и систематически проводимых мероприятий, направленных на удовлетворение заданным требованиям к качеству. Достижение заданных показателей качества ПС в значительной степени зависит от применяемых методов программирования и системы управления качеством.

Многие методы управления качеством программных продуктов [20, 82, 108, 109, 172, 180, 183, 184 и др.] ориентированы либо на количественную оценку готового результата, либо на оценку надежности, получаемую на основе ошибок, фиксируемых при комплексных испытаниях объекта разработки. Методы оценки надежности позволяют установить пригодность или непригодность созданного программного продукта к употреблению.

Узким местом многих методов оценки качества и надежности как одного из свойств является их слабая ориентация на начальные этапы разработки ПС. В [151] предложены прогнозирующие модели надежности, применяемые на этапе программирования и сборки программного продукта. Однако для постепенного достижения требуемого качества требуется применять методы экспертной и количественной оценок промежуточных результатов (состояний) продукта.

Ряд работ посвящен методам экспертной оценки [17, 67, 76 и др.], которые требуют проведения мероприятий по созданию экспертных групп и методик контроля получаемых на этапах признаков продукта, выражаемых качественно или количественно.

В работе [67], в частности, для систем, работающих с базами данных, предлагается уделять большое внимание не только процедурам экспертизы, распределенным по циклу создания проекта, но и психологическим аспектам работы экспертной комиссии, затрагивающей интересы коллективов разработчиков системы.

В результате исследований по управлению качеством ниже будет приведен ряд моделей надежности, которые получили развитие в связи со спецификой предметной области СОД и принятой организацией контроля, сбора статистической информации в ходе разработки программного продукта определенного вида по ТЛ. Предложено на каждом процессе иметь операцию контроля проектных решений, принимаемых при выполнении преобразования состояния объекта разработки.

Работы на следующем процессе начинаются после того, когда результаты предыдущего процесса удовлетворяют требованиям контроля качества.

8.1. ИНЖЕНЕРНЫЕ МЕТОДЫ В РАЗРАБОТКЕ ПС

Разработка ПС как любого нового продукта, с одной стороны носит характер творческой деятельности, а с другой — представляет собой регламентированную последовательность конкретных этапов в изготовлении изделия. Эта двойственность присуща практически всем процессам, сопровождающим создание новых программных средств. Используя традиционные подходы к программированию, разработчику ПО приходится решать как вопросы собственно разработки (исследование предметной области, проектирование изделия, выделение модулей и т. д.), так и вопросы, связанные с управлением отдельными процессами и всей разработкой в целом. При этом, как правило, обе группы вопросов тесно связаны между собой.

Использование различных технологий программирования как раз и предназначено для максимального отделения творческих процессов от работ по управлению разработкой ПС. О качестве программной технологии можно судить по достижению этой цели. Регламентация процесса разработки программ по ТЛ должна сочетаться с экономико-производственными методами управления программированием: планирования трудозатрат в соответствии с ТЛ и наличием специалистов определенной квалификации, учет и контроль выполняемых работ, оценка результатов разработки и т. п. Совокупность методов, используемых для данной цели, будем называть инженерными методами разработки ПС. Они способствуют внедрению в технику программирования промышленных методов разработки программных средств. При этом организация и управление разработкой ПС по ТЛ как раз и привносят в сферу программирования приемы и методы промышленного производства, поскольку ТП отражают не только методы и средства разработки программ, но и методы планирования (по технологическому маршруту) и управления ходом разработки, оценки результатов разработки и деятельности специалистов, выполняющих разработку.

При использовании ТЛ инженер-исполнитель имеет в своем распоряжении карту процесса, где указаны операции для выполнения, требуемые им входные и выходные данные, применяемые методы, средства и инструменты разработки и контроля результатов труда. Одновременно с перечисленными выше факторами рассматриваемые методы включают управляющие воздействия, направленные на соблюдение сроков разработки в соответствии с сетевым графиком и расходами на разработку и т. д.

По мнению многих зарубежных и советских авторов, созданное ПС отображает организационные формы и процессы человеческой деятельности. Поэтому созданные на этапе ТПР технологические линии формализуют результаты деятельности высококвалифицированных специалистов, закладывающих свои знания в процессы регламентированного создания конкретных типов программ, и являются средством для обеспечения технологичности и качества программного продукта.

Инженерные методы основываются на комплекте технологических документов, ведении разработки по конкретной ТЛ, а также инструктивно-методических документах, регламентирующих организацию и управление разработкой ПС высокого качества с применением технологических модулей, реализующих методы разработки. Такой набор документов разрабатывается исходя из потребностей обеспечения инженеров-исполнителей программными и экономическими методами управления разработкой классов программ на основе ТЛ.

8.1.1. ПРИНЦИПЫ ИНЖЕНЕРНОГО ПОДХОДА К РАЗРАБОТКЕ ПС

К основным отнесем принципы инженерного подхода: производственной организации, обеспечения технологичности и качества, планирования трудозатрат.

Принцип производственной организации. В отличие от творческого характера работ, на этапе ТПР при определении ТЛ (анализ ПрО, выбор подходящих средств описания ПрО, корректировка и выбор ограничений на проект и т. д.) инженерные методы, используемые на этапе прикладного программирования, отличаются строгой последовательностью ТП и ТО, оформленных в производственные процессы. В них имеется специализация операций, применяемых методов и инструментов, а также исполнителей, деятельность которых учитывается в соответствии с экономическими методами оценки труда. Необходимым условием успешного использования данного принципа является планирование программных работ, управление которыми осуществляет технологическая служба предприятия, выполняющая контроль соблюдения технологии и оценку объекта разработки, планирование работ с элементами нормирования, совершенствование технологических процессов и др.

В производственных условиях требуется проводить текущее планирование работ, при котором решается задача составления выполнимого (достижимого) плана работ Y по ТЛ. При этом основными данными для составления плана Y являются:

общий плановый срок разработки $[t_0, T]$, где t_0 и T — начальный и конечный сроки разработки;

объем работ $W = \{W_i\}$ с учетом переделок;

требуемые ресурсы $R = \{R_L, R_m\}$, где R_L — людские; R_m — материальные (технические и программные);

нормы потребления людских ресурсов по всем ТП_i $(i = \overline{1, N})$ NR_L и др.

Формально план работ запишется в следующем виде:

$$Y = Y(t_0, T, W, R_L, R_m, NR_L, f), \tag{8.1}$$

где f — случайные факторы (ошибки при выполнении плановых работ на ТП, сбои технических средств и др.), а также факторы, связанные с появлением средств новой техники и программного обеспечения.

В качестве формализованной основы управления разработкой могут использоваться сетевой план-график работ и контроль за его выполнением. В условиях производственной организации ведения разработки на основе ТЛ отсутствие его зачастую приводит к неуправляемому процессу.

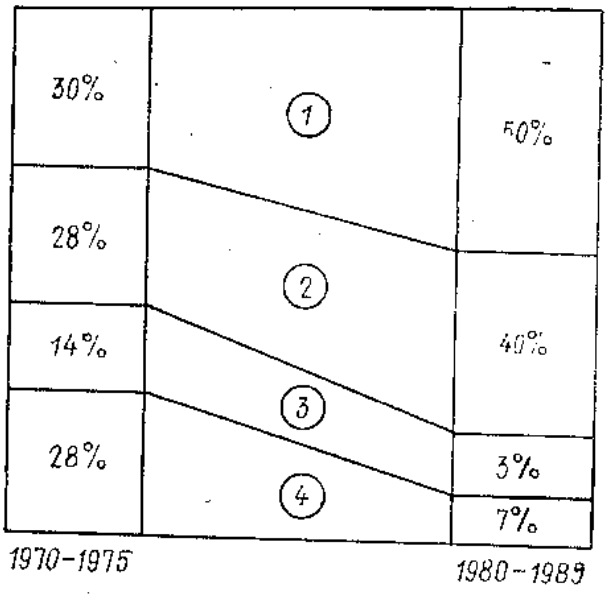


Рис. 8.1. Структура распределения трудозатрат на этапах разработки

Принцип обеспечения технологичности и качества. Понятие технологичности целиком и полностью связано с наличием технологии и с полным соблюдением всех ее требований и правил. Технологичность — это понятие, включающее технологичность ПП и технологичность процесса разработки.

Технологичность ПП — это соответствие ПП потребительским свойствам и определенным функциям ПрО. Обеспечение ее основывается на заложенных в ТЛ элементах типизации, унификации и стандартизации конструктивных элементов ПП, применяемых моделях и заготовках, а также готовых повторно используемых программных объектах из фондов коллективного пользования. Технологичность определяет способность ПП к эксплуатации.

Технологичность разработки — это регламентированность (упорядоченный набор процессов, операций и процедур их выполнения), конструктивность (методом сборки из готового) и инструктивность (методическое и инструктивное обеспечение процессов ТЛ) и организация управления разработкой ПП. Обеспечение ее основывается на применении эффективных методов ведения разработки ПП, воплощенных в ТЛ (или ТП) и направленных на повышение качества и производительности труда, минимизации затрат и времени на разработку ПП.

Принцип планирования трудозатрат. Исходя из результатов исследований [108] основное распределение трудозатрат на этапах разработки (рис. 8.1) приходится на сопровождение и поддержку проекта. Поэтому работы по планированию и нормированию в условиях производственной организации являются актуальной задачей, соответствующей духу времени, по переходу научных разработок на хозрасчет и самофинансирование.

Динамика структуры трудозатрат по основным этапам жизненного цикла ПС в 1970—1985 гг. в США отмечена цифрами в кружках: *1* — сопровождение и поддержка проекта, *2* — анализ и постановки задач, *3* — кодирование и автономная отладка, *4* — комплексная отладка и испытания. Данная структура осталась на период 1985—1990 гг.

Взятый за основу подход разработки программ по ТЛ позволяет определить трудозатраты (в человеко-днях) на разработку программ при применении ТЛ исходя из следующих исходных данных:

1) N — количество процессов на ТЛ;

2) J_i — количество операций на ТП_i $(i = \overline{1, N})$;

3) $\sum_{i=1}^{N} P_i + P$ — директивная продолжительность разработки всей программной системы, где P_i — минимально требуемая продолжительность (в днях) для i-го процесса; P — резерв времени, который остается до планового срока и используется для варьирования исходными продолжительностями процессов;

4) $x_{ij}\,(i = \overline{1, N},\ j = \overline{1, J_i})$ — объем ПС (в операторах), задаваемый экспертной оценкой. ПС должно разрабатываться на i-м процессе и j-й операции;

5) T_{ij} — производительность труда (в операторах в день) при выполнении j-й операции на процессе i.

За искомые величины примем: γ_i — оптимальная продолжительность i-го процесса, m_{ij} — количество программистов, необходимых для выполнения j-й операции на i-м процессе. Тогда $\sum_{j=1}^{J_i} m_{ij}$ задает количество исполнителей, необходимых для разработки ПС на i-м процессе, а $F = \max \sum_{i=\overline{1, N}} \sum_{j=1}^{J_i} m_{ij}$ — максимальное количество специалистов, необходимых для реализации всей прикладной системы на основе ТЛ.

Исходя из введенных обозначений математическую модель задачи планирования разработки ПС сформулируем следующим образом:

$$F^0 = \max \sum_{i=\overline{1, N}} \sum_{j=1}^{J_i} m_{ij} \Rightarrow \min,$$

$$x_{ij} \leqslant \gamma_i m_{ij} T_{ij},\quad \gamma_i \geqslant P_i, \tag{8.2}$$

$$\sum_{i=1}^{N} \gamma_i \leqslant \sum_{i=1}^{N} P_i + P. \tag{8.3}$$

Величины x_{ij}, T_{ij}, P_i, P известны, а γ_i и m_{ij} являются целевыми переменными. Ограничения на m_{ij} выражаются в виде $m_{ij} \geqslant 1$.

Формула (8.2) означает, что x_{ij} — количество операторов — в ПП должно быть создано m_{ij} специалистами за γ_i дней, а неравенство (8.3) означает, что разработка ПП должна быть выполнена в плановый срок.

Задача имеет много вариантов с большим перебором, требующих некомбинаторных методов решения, поэтому перейдем от этой математической модели к задаче линейного программирования путем изменения критериев и линеаризации ограничения (8.2). Для этого в

F^0 потребуем приведения всех сумм $\sum_{j=1}^{J_t} m_{ij}$ к минимально возможной величине. В этом случае

$$F^0 = d_1 \sum_{i=1}^{N} \left(\sum_{j=1}^{J_t} m_{ij} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{J_i} m_{ij}}{N} \right) + \\ + d_2 \sum_{i=1}^{N} \sum_{j=1}^{J_i} m_{ij} \Rightarrow \min, \qquad (8.4)$$

где d_1, d_2 — управляющие параметры.

Введем дополнительные переменные $y_i \geqslant 0$ и $z_i \geqslant 0$ такие, что

$$y_i - z_i = \sum_{j=1}^{J_t} m_{ij} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{J_t} m_{ij}}{N}.$$

Тогда пролинеаризуем неравенство (8.2):

$$x_{ij} - \gamma_i^0 m_{ij} T_{ij} - m_{ij}^0 \gamma_i T_{ij} \leqslant -\gamma_i^0 m_{ij}^0 T_{ij}. \qquad (8.5)$$

Здесь γ^0, m_{ij}^0 соответствуют начальным приближениям, γ_i^0 — управляющим параметрам, которые выражены следующим соотношением:

$$m_{ij}^0 = \frac{x_{ij}}{\gamma_i^0 T_{ij}}.$$

Исходя из этих предположений получаем задачу линейного программирования

$$F^0 = d_1 \sum_{i=1}^{N} \left(\sum_{j=1}^{J_t} m_{ij} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{J_i} m_{ij}}{N} \right) + d_2 \sum_{i=1}^{N} \sum_{j=1}^{J_t} m_{ij} \Rightarrow \min,$$

$$x_{ij} - \gamma_i^0 m_{ij} T_{ij} - m_{ij}^0 \gamma_i T_{ij} \leqslant -\gamma_i^0 m_{ij} T_{ij}, \ \gamma_i \geqslant P_i, \qquad (8.6)$$

$$\sum_{i=1}^{N} \gamma_i \leqslant \sum_{i=1}^{N} P_i + P,$$

$$m_{ij}^0 = \frac{x_{ij}}{\gamma_i^0 T_{ij}}, \ m_{ij} \geqslant 1,$$

$$\frac{\sum_{i=1}^{N} \sum_{j=1}^{J_i} m_{ij}}{N} - \sum_{j=1}^{J_k} m_{ij} - y_i + z_i = 0.$$

При решении этой задачи будут получены γ_i, m_{ij} (нецелочисленные), с помощью которых определяются целочисленные значения базовой модели, т. е. будет получен пробный вариант решения задачи планирования трудозатрат.

Варьируя значениями управляющих параметров d_1, d_2 и γ_i^0, получим окончательное решение задачи планирования, т. е. искомое число специалистов, необходимых для разработки в срок программной системы по заданной технологической линии.

8.1.2. ОРГАНИЗАЦИЯ И УПРАВЛЕНИЕ РАЗРАБОТКОЙ ПРОГРАММ ПО ТЛ

Управление разработкой программ по ТЛ включает решение вопросов планирования хода разработки, учет и контроль.

Метод планирования требуемого числа специалистов для проведения разработки описан выше. Здесь рассмотрим средства, способствующие планированию конкретных работ и учету их выполнения. Введем в рассмотрение специальные информационные структуры — карты выполнения работ, разрабатываемые документы, сроки начала и окончания выполнения отдельных операций и квалификацию исполнителя. Такие карты ведутся ответственным исполнителем процесса разработки и могут контролироваться представителем технологической службы. Они создаются на основе технологического маршрута и сетевого плана-графика.

Одним из сложных моментов планирования работ является определение норм на каждого специалиста и на каждый процесс. Специалист должен иметь определенный квалификационный уровень (Q) и обладать навыками работ в соответствии со спецификацией процесса (проектирование баз данных, программирование вычислительных модулей и др.).

На ТП, связанных с реализацией программ, трудоемкость на одного специалиста (исходя из данных, приведенных в итоговом отчете НИР-86) Советско-Болгарского научного объединения «Интерпрограмма» вычисляется по формуле

$$A = \left(\frac{1 + (I)^{0,8}}{c\sqrt{QT}} + \frac{D}{d}\right)\frac{R}{B}, \tag{8.7}$$

где I — число операторов; D — число страниц документации; Q — коэффициент квалификации (не более 1); T — коэффициент технологической обеспеченности операции (не более 1); R — коэффициент трудоемкости применения процесса; $b = 21{,}5$ — константа (средняя) числа дней в месяце; $c = 30$ — число операторов в день (среднее); $d = 2$ — число страниц в день (усредненное).

Коэффициент квалификации в формуле (8.7) может быть вычислен по формуле

$$Q = \frac{1}{n}\sum_{m=1}^{n}\frac{1}{m}\sum_{i=1}^{m} q_i\ (i), \tag{8.8}$$

где q_i — коэффициент квалификации i-го специалиста в определенном квартале; n — число кварталов; m — число участвующих в разработке ПС.

Коэффициент квалификации определяется экспертным путем и выражает отношение производительности труда данного специалиста к средней производительности для категории разработчиков, к которой он относится.

Коэффициент T в формуле (8.7) предложено вычислять по формуле

$$T = \frac{T_l + T_M + T_n + T_c}{4},$$

где T_l — коэффициент применяемого языка ($T_l = 0{,}6$ для Ассемблера; 0,7 — для макроассемблера; 0,8 — для ЯП высокого уровня; 1,0 — современных ЯП типа Паскаль, АДА и др.); T_M — коэффициент применяемости программных методов на ТЛ (если метод не применяется, то $T_M = 0{,}5$, в противном случае — $T_M = 0{,}8$); T_n — коэффициент применяемости инструментальных средств ($T_n = 0{,}8$ — традиционные методы и $T_n = 1{,}0$ — новые); T_c — коэффициент операционной среды (определяется на основе экспертных оценок, учитывающих уровень технологичности операционной среды).

Учет выполнения разработки по ТЛ на основании графика выполнения работ проводится для получения информации о состоянии работ за определенный промежуток времени и использовании ее для контроля и регулирования с целью достижения заданных показателей качества.

В современных условиях, когда требуется высокое качество выпускаемой продукции, учет информации и контроль хода разработки являются необходимым условием обеспечения промышленного способа разработки.

Контроль за ходом выполнения разработки ПП проводится для выявления отклонений фактических показателей от плановых и формирования определенной информации о характере и причинах отклонений. Результаты работ, подвергающиеся контролю, представляют собой карты выполнения работ, ПТД и программы, формируемые в процессе разработки. Результаты контроля оформляются протоколом, являющимся основанием для дальнейших доработки и разработки.

Операция контроля, предусматриваемая как заключительная на каждом ТП, регламентирует вид документов, подлежащих контролю, и результат. Контроль проводится специальной группой, входящей в состав технологической службы. Руководитель этой группы должен иметь высокую квалификацию и знание объекта контроля. В состав группы входят ответственный (старший) технолог и контролеры техпроцессов ТЛ. Кроме того, в эту группу могут входить и уполномоченные по стандартизации и нормоконтролю выпускаемой документации.

Содержательный контроль включает технологический контроль за соблюдением технологической дисциплины (полнота и корректность ПТД), правил и условий выполнения каждой операции (точности применения методов, средств и инструментов), а также нормативов, установленных в соответствующих технологических документах и применяемых на ТЛ.

Контроль технологичности и качества, проводимый контролерами ТП, и на заключительном этапе совместно со старшим технологом осуществляется на основе методов, которые будут описываться в дальнейшем изложении.

8.1.3. МЕТОДОЛОГИЧЕСКАЯ ПОДДЕРЖКА ТЕХНОЛОГИИ

Эффективность применения ТЛ зависит от степени их инструктивности. Последние определяются технологическими документами (ТД) общего и частного характера, присущими конкретному процессу. В них определены регламентация и порядок применения инструментальных средств, форм представления результатов выполнения процессов и документирования продукта, а также управления работами по получению программного продукта высокого качества.

Документы поддерживают единый подход к созданию ПС по ТЛ с использованием стандартных форм документов для фиксации проектных решений на всех этапах разработки. Для них принят стандарт в соответствии с требованиями ГОСТ ЕСКД. Дадим краткую характеристику ТД, применяемых в системах обработки данных (СОД).

1. ТД «Язык проектирования» определяет методику применения этого языка (типа PDL) для описания структур программ и способствует улучшению качества проектирования и единообразному представлению проектных решений; составлению ЭПД на ПС; формализации методов контроля разработки ПС и т. д.

2. Набор ТД «Методика применения ТМ» определяет способы использования инструментальных средств: различных СУБД, пакетов программ для ввода, контроля, обработки данных. Эти документы способствуют освоению и использованию основанных на них методов и средств выполнения соответствующих технологических операций (процессов) ТЛ.

3. ТД «Организация и контроль разработки ПС» определяет задачи и функции группы контроля качества ПС по проведению контроля хода разработки. Содержит описание видов и методов контроля, порядок формирования номенклатуры показателей качества и методические указания по систематической экспертной или количественной оценке показателей качества, достигнутых в ходе разработки ПС на отдельных процессах.

Целью данного документа является: формализация методов анализа состояния разработки; выявление отклонений показателей качества ПС от заданного уровня и нарушений требований и сроков разработки в соответствии с планом-графиком работ. Описаны принципы организации коллектива разработчиков для создания ПС по ТЛ исходя из оргструктуры, заложенной в каждой ТЛ.

4. ТД «Методика тестирования ПС» содержит рекомендации по организации и проведению тестирования на разных этапах процесса разработки ПС (автономное и комплексное тестирование), определяет порядок составления и содержание плана тестирования, фиксации обнаруженных ошибок в журналах регистрации ошибок и методы оценки полноты тестирования по разным критериям. Методика нацелена на

тщательное планирование процесса тестирования и проведение оценки его результатов.

5. ТД «Формы проектно-технологических документов, используемых при разработке ПС» представлен набором модулей табличного вида. Даны их назначение и порядок заполнения при выполнении ТО, приведены комплекты ПТД, рекомендуемые для разных видов объектов (программный компонент, программный модуль) и поддерживающие разные процессы (программирование, тестирование на основе таблиц тестов и сбор статистики об ошибках в журналах регистрации ошибок и т. д.).

6. ТД «Порядок описания прикладных модулей. Стандарт модулей» предлагает стандартизованную структуру прикладного модуля, удобную для разработки и составления документации. В классе рассматриваемого ППО СОД определены типы модульных элементов: функциональные заготовки, прикладные и системные модули. Для них предложена рекомендуемая стандартизованная форма представления, включающая описание внешних спецификаций и текста модуля.

7. ТД «Описание текста модуля» определяет структурное оформление текста модуля в любом ЯП и обеспечивает унификацию документов ЕСПД.

Рассмотренный набор технологических методик и инструкций может быть использован для регламентации работ при разработке программных средств других классов. Они могут уточняться и дополняться исходя из условий разработки и используемых инструментальных средств.

8.2. МЕТРИЧЕСКИЕ ОЦЕНКИ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ РАЗРАБОТКИ ПС

Процесс проектирования ПС ориентирован на повышение производительности труда разработчиков (количество строк, команд в день) за счет четкой регламентации и специализации операций каждого процесса, входящего в ТЛ.

Повышение производительности и достижение требуемого качества ПС зависят от:

правильности и полноты методов и средств, выбранных для реализации данного вида ПС и представленных упорядоченной последовательностью (исходное качество процесса) на ТЛ;

контроля соблюдения технологии ТЛ и результатов изменения состояний объекта для определения степени отклонения от регламентных ТМ и средств, а также для определения погрешностей отклонения отдельных свойств текущего состояния объекта от установленных метрик и определения максимума погрешности на конечном этапе жизненного цикла как степени несоответствия объекта разработки требованиям заказчика;

унификации и стандартизации конструктивных элементов объекта разработки (функциональные заготовки и готовые модули) и их качества;

спецификации и квалификации субъектов (умение программиста работать на инструментальных средствах и др.), автоматизирующих выполнение операций ТЛ;

организации и управления разработкой (планирование норм, учет, контроль хода разработки и др.).

Поскольку качество получаемого ПС определяется в значительной степени указанными факторами, а также применяемыми методами изменения характеристик процесса и состояний программного объекта на процессах ТЛ, то далее в работе рассматривается аналитическая модель процесса, основанная на учете влияния сложности объекта разработки на качество процесса проектирования и уровня вносимых субъектом разработки ошибок в объект разработки.

8.2.1. УЧЕТ ВЛИЯНИЯ СЛОЖНОСТИ ПРОГРАММНОГО ОБЪЕКТА НА КАЧЕСТВО ПРОЦЕССА РАЗРАБОТКИ ПС

Сложность программного продукта является глобальной характеристикой ТЛ и должна контролироваться и измеряться в процессе всего хода разработки.

Меры свойств процесса (трудоемкость, производительность и др.) являются измеряемыми, хотя полностью не отражают реального состояния объекта разработки. Меры свойств объектов разработки (количество спецификаций, данных, операторов и т. п.) характеризуют состояние развиваемого объекта в заданный момент времени. Они зависят от сложности создаваемого продукта, а, следовательно, и от процесса его реализации. Кроме того, в тесной связи с этими факторами находятся квалификация и состав исполнителей, производящих разработку.

Рассмотрим понятие состояния разработки S в конкретный момент времени. Оно определяет набор средств для решения поставленной задачи, находящихся на определенном уровне детализации и конкретизации. Элементами состояния программного объекта могут быть: спецификации; алгоритмы решения задачи; описание алгоритма в языке программирования; программные средства как составные элементы различной степени готовности; техническая и технологическая документация; набор тестов; контрольные примеры и т. д.

Начальное состояние программного объекта для решаемой задачи состоит из общих спецификаций, выраженных в виде технического задания или других форм описания требований. Конечному состоянию соответствуют готовое программное средство с необходимым комплектом технической и технологической документации и контрольные примеры проверки его работоспособности. Переход из одного состояния в другое связан с постепенным уточнением начальной постановки задачи, выделением отдельных функций и соответствующих программных компонент, конкретизацией информационных структур и т. д.

Технология ведения разработки объекта с учетом его состояний в ТП отражает процесс развития состояний S, приводящий к последовательному его усложнению. Сложность программного продукта является его внутренним свойством и определяется целевым назначением —

показателем сложности Δ, характеризующим взаимодействие между программным объектом и субъектом разработки. Показатель сложности является характеристикой, точная оценка которого представляет значительные трудности [154]. Поэтому будем его рассматривать как суммарную сложность составляющих элементов объекта разработки.

Объект разработки при переходе к новому состоянию может претерпевать следующие изменения:

подвергаться делению на более мелкие элементы (деление систем на подсистемы, программ на модули и т. д.);

переходить в качественно новое состояние или изменять свои количественные характеристики (переход от описания модуля на языке спецификаций к языку программирования, трансляция исходного текста с получением объектного модуля и т. д.);

сопровождаться разработкой дополнительных сопутствующих элементов (документации, тестов, контрольных примеров и т. д.).

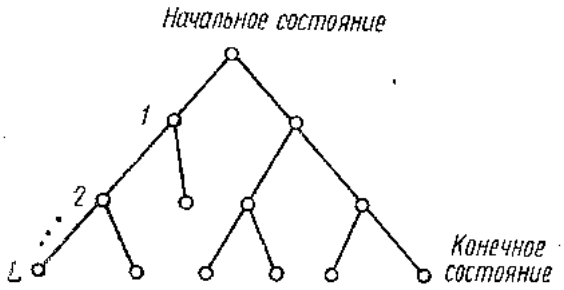
Нетрудно заметить, что все эти изменения связаны с увеличением количества элементов разработки. В первом случае число элементов возрастает за счет разбиения, во втором — увеличения количества представлений отдельного элемента, в третьем — создания дополнительных элементов. Тогда процесс развития состояния S можно представить в виде дерева, приведенного на рис. 8.2.

Из каждой вершины в общем может исходить несколько ребер. В дальнейшем будем рассматривать только бинарные деревья. Покажем, что любое изменение элемента разработки может быть представлено в виде бинарного дерева.

1. Деление объекта разработки. Допустим, объект можно разделить на k более мелких элементов. Этот процесс может быть представлен в виде последовательности состояний $S_1, S_2, \ldots, S_{k-1}$, где S_i связан с выделением i-го элемента. Соответствующее дерево представлено на рис. 8.3.

2. Изменение качественного состояния или количественных характеристик. Данная ситуация представлена на рис. 8.4.

Необходимость сохранять старый элемент объекта связана с возможными ошибками, возникающими в ходе разработки, и возвратом к предыдущему элементу объекта.



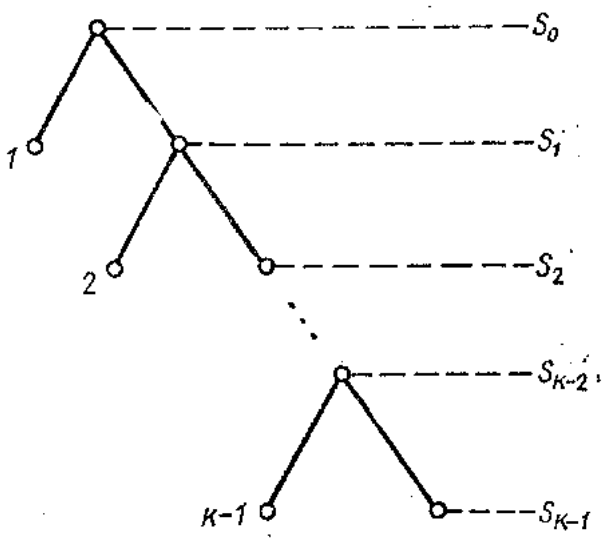


Рис. 8.2. Дерево развития состояния программного объекта

Рис. 8.3. Развитие состояния для операций деления элемента объекта разработки

3. Разработка дополнительных элементов. Пусть для объекта требуется разработать k дополнительных элементов. Тогда процесс описывается последовательностью состояний S_1, S_2, ..., S_k. Соответствующее дерево представлено на рис. 8.5. В результате такого представления дерево развития состояния будет бинарным и для его анализа можно использовать соответствующий математический аппарат. В бинарном графе с N вершинами содержится N—1 дуг. Если каждой дуге сопоставить технологическую операцию, то полное число операций не будет превышать N — 1 (некоторые операции представлены формально и не требуют никаких действий: например, исходный и старый элементы на рис. 8.4 соответствуют одному и тому же объекту).

Пусть $\bar{\delta}$ — средняя сложность отдельного элемента объекта, а $\bar{\tau}$ — средняя длительность одной технологической операции его преобразования. Тогда сложность всей разработки Δ приблизительно выразится следующим равенством:

$$\Delta = N\bar{\delta}. \tag{8.9}$$

Общее время разработки приблизительно определяется выражением

$$T = (N - 1)\bar{\tau}. \tag{8.10}$$

Из (8.9) и (8.10) получаем зависимость между сложностью разработки и ее общим временем:

$$\Delta = \frac{T + \bar{\tau}}{\bar{\tau}}\bar{\delta}. \tag{8.11}$$

Отсюда средняя длительность одной технологической операции

$$\bar{\tau} = \frac{T\bar{\delta}}{\Delta - \bar{\delta}}. \tag{8.12}$$

В предыдущем анализе использовались средние значения $\bar{\delta}$ и $\bar{\tau}$. В соответствии с этим предполагалось, что любая технологическая опе-

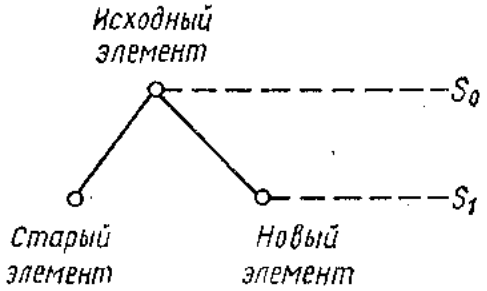


Рис. 8.4. Развитие состояния для изменения отдельного элемента объекта

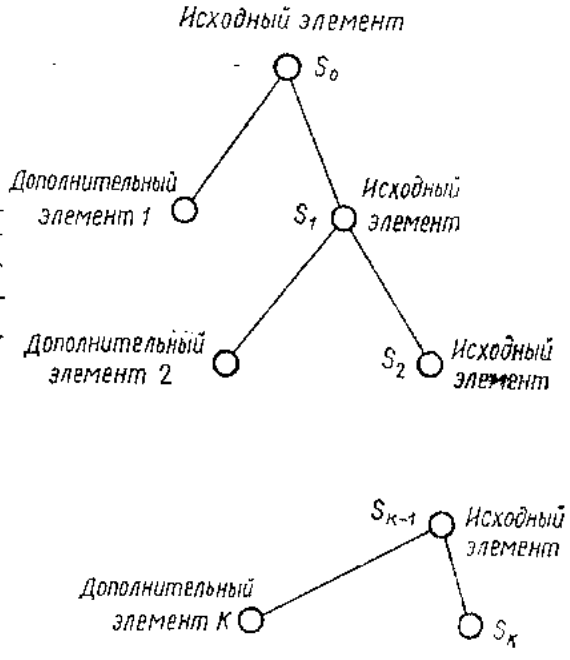


Рис. 8.5. Развитие состояния объекта для случая разработки дополнительных элементов

рация для любого из соответствующего множества элементов объекта разработки может быть выполнена любым специалистом, входящим в коллектив разработчиков. В реальной ситуации необходимо учитывать обобщенности объектов разработки и технологических операций, а также квалификацию исполнителей. Пусть квалификация исполнителя оценивается некоторым коэффициентом q_r, δ_{i-1} — сложность исходного элемента, δ_i — сложность элемента, полученного в результате применения технологической операции $ТО_i$.

Тогда $\tau_i = \tau_i(\delta_{i-1}, ТО_i, q_r)$, $\delta_i = \delta_i(\delta_{i-1}, ТО_i, q_r)$ и

$$T = \sum_{i=1}^{N} \tau_i, \tag{8.13}$$

$$\Delta = \sum_{i=0}^{N} \delta_i. \tag{8.14}$$

Качество программного продукта будет функцией от τ_i, δ_i и N. В предположении, что общая сложность Δ является объективной характеристикой и не зависит от конкретных исполнителей ($\Delta = \text{const}$), следует, что показатель качества будет меняться в зависимости от значений τ_i, δ_i и N или от δ_0, $ТО_i$, q_r, где δ_0 — сложность начального состояния объекта разработки (спецификации).

Разработка программного продукта согласно ТЛ позволяет повысить его качество за счет следующих факторов.

1. Количество технологических процессов и операций фиксировано, что позволяет разбить все $ТО_i$ на классы однотипных операций.

2. Выделение классов операций позволяет упростить зависимость числа элементов разработки: на одной и той же ТЛ $N = N(\delta_0)$, в то время как в общем случае $N = N(\delta_0, ТО_i)$.

3. Каждая технологическая операция определяется требованием к квалификации исполнителя q_r. Это позволяет подбирать состав исполнителей и распределять между ними обязанности согласно их квалификации.

4. При заданном времени разработки T и известном составе исполнителей можно так выбрать τ_i и δ_i, чтобы показатель качества был максимальным. В частности, это позволяет дать ответ на вопрос: возможно ли решение поставленной задачи δ_0 за определенное время T при известном составе исполнителей с требуемым показателем качества.

8.2.2. ОПРЕДЕЛЕНИЕ ЗАТРАТ НА РАЗРАБОТКУ ПРОГРАММ НА ОСНОВЕ ТЛ

Рассмотрим вопрос определения затрат на примере конкретных функционально-ориентированных ТЛ. Каждая линия включает этапы предпроектных и проектных исследований, являющихся творческой частью работ. Затраты на их ведение (методика и форма фиксации результатов исследований приводится в описаниях ТЛ) не могут оцениваться количественно из-за неопределенности единиц измерения наиболее творческой части программистской деятельности.

На остальных процессах ТЛ могут применяться оценки затрат на разработку, связанные, например, с такими единицами измерения, как количество строк, операторов и т. п., разрабатываемых в ходе ведения работ, учитываемых и контролируемых в базе проекта.

На ТЛ одним из процессов является процесс сборки, основанный на применении метода сборочного программирования. При этом непосредственные затраты идут на создание новых компонент, на анализ и комплектацию готовых (повторно используемых) программных объектов и на определение «цены модульности» при сборке программных компонент. В связи с этим в расчет производительности труда входят затраты для вновь создаваемых объектов по ТЛ и затраты на интерфейс готовых компонент.

Более подробно рассмотрим задачу минимизации затрат на определение интерфейса объединяемых модулей по методу сборки.

С этой целью рассмотрим вначале задачу оценки вариантов компоновки структуры программы P по исходному графу. Применяя метод упорядоченного перебора вариантов связи модулей по данным (экспортируемым и импортируемым) на основе анализа операторов вызова, определим коэффициент $K^p_{\text{ин}}$ информационной независимости программы P:

$$K^p_{\text{ин}} = \sum_{l=1}^{N} \xi_{n_l} / \xi_p, \tag{8.15}$$

где ξ_{n_l} — мера связанности по данным l-модуля; ξ_p — мера связанности по всем данным программы P и определяется как суммарная сложность описания межмодульных интерфейсов модулей, входящих в P:

$$\xi_P = \xi_n - \sum_{l=1}^{n} \xi_{n_l}.$$

«Цена модульности» создаваемой программы P, т. е. объем интерфейса $V_{\text{доп}}$ (в строках или командах Ассемблера) создаваемых интерфейсных модулей связи для N разноязыковых модулей, вычисляется по формуле

$$V_{\text{доп}} = \sum_{l=1}^{N/2} d(M'_{jk})\, \xi_{n_l}, \tag{8.16}$$

где $d(M'_{jk})$ — длина модуля связи для M_j и M_k.

Объем создаваемой программы

$$V_p = (V_{\text{доп}} + \sum_{l=1}^{N/2} d(M^l_L)\, K^P_{\text{ин}})\, \Delta V, \tag{8.17}$$

где $\Delta V = \dfrac{V - V_{\text{доп}}}{N}$ — коэффициент отклонения проектного объема V от полученного; $d(M^l_L)$ — длина l-модуля в одном языке L.

Зная объем $V_{\text{доп}}$ и число исполнителей на операции сборки, можно подсчитать трудоемкость выполнения этой операции. В общие затраты

на компоновку модулей входят также затраты на определение необходимого состава модулей, их функциональной пригодности и соответствие входных данных структурам данных реализуемой ПрО.

Суммарные стоимостные затраты сборочного создания агрегатов определим следующим образом:

$$C = C_{аз} + C_{зм} + C_{сб}, \tag{8.18}$$

где $C_{аз}$ — затраты на анализ задачи Пр0; $C_{зм}$ — на анализ возможностей использования готовых объектов для реализации задач Про; $C_{сб}$ — на сборку комплекта модулей.

Затраты на анализ задач

$$C_{аз} = \sum_{r=1}^{N} b_r C_1(D_r), \tag{8.19}$$

где $C_1(D_r)$ — затраты на анализ данных r-задачи Пр0 ($r = \overline{1, N}$ — число задач); $b_r = \begin{cases} 1, \text{ если } r\text{-задача выбрана для реализации функции Пр0,} \\ 0 \text{ — в противном случае.} \end{cases}$

Определим затраты на анализ возможностей использования имеющегося фонда функциональных модулей для реализации требуемых задач СОД (АСУ):

$$C_{зм} = \sum_{j=1}^{N} \sum_{r=1}^{N_1} a_{rj} C_2(M_{jr}), \tag{8.20}$$

где $C_2(M_{jr})$ — затраты на анализ функциональных возможностей j-модуля для решения r-задачи ($j = \overline{1, N}$ — число модулей); $a_{rj} = \begin{cases} 1, \text{ если } j\text{-модуль используется для выполнения } r\text{-задачи,} \\ 0 \text{ — в противном случае.} \end{cases}$

Затраты на сборку модулей определяются следующим образом:

$$C_{сб} = \sum_{j=1}^{N} \sum_{r=1}^{N_1} \sum_{k=1}^{Nr} d_{jrk} C_3(V(M'_{jr})), \tag{8.21}$$

где $C_3(V(M'_{jr}))$ — в которой $V(M'_{jr}) = V_{доп}$ — затраты на сборку M_j- и M_r-модулей; $d_{jrk} = \begin{cases} 1, \text{ если } k\text{-параметр из набора } x = \{x_1, \ldots, x_l\} \\ \text{является входным для } j\text{-модуля } r\text{-задачи } (l = \\ = \overline{1, N_2} \text{ — число параметров),} \\ 0 \text{ — в противном случае.} \end{cases}$

Таким образом, формула (8.20) приобретает вид

$$C = \sum_{r=1}^{N} b_r C_1(D_r) + \sum_{j=1}^{N} \sum_{r=1}^{N_1} a_{rj} C_2(M_{jr}) + \\ + \sum_{j=1}^{N} \sum_{r=1}^{N_1} \sum_{k=1}^{N_2} d_{jrk} C_3(V(M'_{jk})). \tag{8.22}$$

Основными ограничениями данного выражения являются:

1) необходимость реализации требуемых функциональных задач Пр0

$$\sum_{r=1}^{N} b_r \geqslant N_0, \tag{8.23}$$

где N_0 — число реализуемых функциональных задач;

2) совместимость выбираемого набора модулей с языковыми средствами среды программирования

$$a_{rj}\rho_{jL} = \rho'_L, \tag{8.24}$$

где $\rho_{jL} = 1$, если j-модуль описан в ЯП высокого уровня, и $\rho_{jl} = 0$ — в противном случае; $\rho'_l = 1$, если в среде программирования имеется система программирования в языке L, и $\rho'_L = 0$ — в противном случае;

3) решение r-задачи за время T_r

$$\sum_{j=1}^{N_j} a_{rj}\tau_r \leqslant T_r. \tag{8.25}$$

4) объем, необходимый для реализации агрегата:

$$\sum_{l=1}^{l} V_l \leqslant V_{\max}, \tag{8.26}$$

где $V_{\max}$ — максимальный объем допустимой памяти для загрузки в нее сформированного агрегата.

Расчет стоимостных затрат, выраженных формулой (8.22), является трудоемким процессом, поэтому необходимо минимизировать каждую ее составляющую.

8.3. УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОГРАММ ПО ТЛ

Под **управлением качества** ПС будем понимать действия (операции), осуществляемые в ходе разработки ПС по технологическим процессам с целью установления, обеспечения и поддержки необходимого уровня качества.

Формой установки качества является контроль по $M_{\text{кач}}$ показателей свойств создаваемого ПС на процессах ТЛ для получения информации о фактическом состоянии объекта посредством экспертизы проектной документации, анализа правильности выходных данных на этапе отладки (тестирования) отдельных модулей или ПС в целом.

Полученная информация сопоставляется с заранее заданной в техническом задании на ПС и в $M_{\text{кач}}$. Результат сопоставления используется для выдачи заключения о текущем состоянии объекта разработки при определении соответствующих видов управляющих воздействий на объект или средства труда. Вид воздействия зависит от причины отклонения от требуемого уровня качества и способа ее устране-

ния. Суть воздействия состоит в качественном изменении состояния объекта.

Оценка качества — это количественное определение значений показателей свойств как создаваемого программного объекта, так и инструментов, поддерживающих процесс его разработки (трансляторов, редакторов, отладчиков и т. п.), на основе ТЛ.

Основная цель оценки качества в ходе разработки состоит в проверке правильности выполнения операций ТП, выявлении отклонений фактических показателей качества от плановых, а также сборе и формировании текущей информации о характере и причинах отклонений.

В табл. 8.1 приведены задачи управления качества, ранжируемые по процессам общего назначения.

Таблица 8.1

Наименование этапа (процесса)	Решаемые задачи
Разработка ТЗ и спецификаций требований	Разработка моделей качества
Разработка ПС	Управление качеством. Сравнение фактических показателей качества с проектными
Испытание ПС	Оценка полученных и эксплуатационных показателей. Экспертиза технической документации. Оценка научно-технического уровня ПС
Изготовление ПС	Оценка качества изготовления ПС и документации
Эксплуатация ПС	Оценка эксплуатационных показателей качества. Заключение о соответствии исходным требованиям
Сопровождение ПС	Оценка удобства сопровождения ПС

8.3.1. МОДЕЛЬ КАЧЕСТВА И КОНТРОЛЬ ПОКАЗАТЕЛЕЙ КАЧЕСТВА В ХОДЕ РАЗРАБОТКИ ПС

Модель качества разрабатывается на этапе ТПР при создании ТЗ на проектируемое ПС. Для модели проводятся контроль показателей в ходе разработки и количественная оценка. Рассмотрим показатели свойств (факторы) надежности более конкретно.

Показатели свойств назначения. В этом показателе оценочный показатель *сложность* характеризуется объемом, выраженным количеством операторов исходного текста:

$$V = \sum_{i=1}^{N} (k \times O[i]), \qquad (8.27)$$

где $O[i]$ — количество операторов в i-модуле; k — уровень ЯП ($k = 1$ для Ассемблера и $k = 10$ для ЯП высокого уровня).

Полнота реализации функций характеризует степень удовлетворения требований пользователя к функциям и определяется по двум оценочным показателям:

1) коэффициент полноты реализованных функций

$$K_n = F/P, \tag{8.28}$$

где P — число функций, включенных в требования к ПС; $F = \sum_{i=1}^{P} (F_i)$ — общее число реализованных функций; $F_i = \begin{cases} 1, \text{ если функция реализована,} \\ 0 \text{ — в противном случае.} \end{cases}$

2) средний взвешенный показатель полноты реализованных функций

$$\Phi = \sum_{i=1}^{P} (W_i * F_i), \tag{8.29}$$

где W_i — коэффициент весомости функции F_i — определяется из условия, что $\sum_{i=1}^{P} W_i = 1$,

Показатель *защищенности* (Π_3) информации определяется из соотношения $\Pi_3 = V_3/V_d$, где V_3 — объем данных, фактически защищенных; V_d — требующих защиты.

Показатели свойств функционирования. Основной показатель этого свойства — *корректность.* Он определяется степенью соответствия ПС требованиям пользователя, и им можно управлять в ходе разработки. Методами определения корректности (контроль проектных решений, тестирование) достигается снижение числа ошибок путем применения на операциях процессов формальных методов проектирования (SA-диаграммы, HIPO-схемы, *P*-схемы и др.).

Оценка *надежности* функционирования связана с оценкой корректности и проводится на основании собранной статистики по результатам тестирования и испытаний, накапливаемых в журнале регистрации ошибок (таб. 8.7 — 8.9) с помощью моделей.

Надежность функционирования определяется двумя показателями.

1. Безотказность определяется средним временем наработки на отказ.

О т к а з — это процесс обнаружения и регистрации ошибок, требующих определения причины возникновения ошибок. Под о ш и б к о й будем понимать причину, по которой результат функционирования программы не адекватен требуемому.

2. Восстанавливаемость определяется средним временем обнаружения и устранения ошибки. В качестве числовой характеристики используется коэффициент готовности

$$K_г = T_{ср}/(T_{ср} + T_в), \tag{8.30}$$

где $T_{ср}$ — среднее время наработки на отказ; $T_в = T_о + T_л$ — среднее время восстановления; $T_о$ — среднее время обнаружения ошибки; $T_л$ — время локализации ошибки.

Показатели эргономичности характеризуют удобство эксплуатации и оцениваются в процессе эксплуатации. Учет этих показателей в ходе разработки заключается в проверке наличия средств переносимости, обучения, диалога с подсказкой и др.

Переносимость определяется по формуле

$$П = З_а/З_п, \quad (8.31)$$

где $З_а$ — затраты на адаптацию в новой операционной среде; $З_п$ — затраты на полную разработку в новой операционной среде.

Обучаемость определяется удобством построения диалога и наличием обучающих курсов:

$$Y_д = О_п/N, \quad (8.32)$$

где $О_п$ — количество операторов с подсказкой; N — общее количество операторов.

Простота подготовки к работе определяется количеством технологических операций при подготовке ПС к работе.

Документируемость определяется полнотой документации (по составу), простотой изложения, структурностью информации. Оценивается при экспертизе группой нормоконтроля, носит качественный характер и оформляется протоколом.

Показатели свойств технологичности характеризуются рациональным выделением и использованием типовых, унифицированных компонент, способствующих сокращению трудовых затрат на создание ПС по ТЛ.

Уровень унификации определяется коэффициентами применяемости ($K_{пр}$) и повторяемости ($K_{пов}$), которые вычисляются по формулам

$$K_{пр} = (N - N_0) * N, \quad (8.33)$$

$$K_{пов} = N_1/N,$$

где N — общее количество составных элементов в формируемом ПС; N_0 — количество оригинальных элементов; N_1 — количество повторно используемых элементов (модулей).

При управлении качеством на основе ТЛ для рассмотренных свойств показателей основными факторами, влияющими на качество на этапах проектирования, являются методики, инструкции и контроль хода разработки. Факторами, влияющими на эксплуатационные характеристики, являются: корректность постановки задач и исходных требований; полнота тестирования; полнота и точность отображения результатов в ПТД.

Контроль качества проектирования ПС по ТЛ. В процессе разработки контролю на полноту подвергаются функциональная ФА и системная СА архитектуры. Контроль является качественным и проводится экспертным путем на основании требований к функциям. Результат контроля фиксируется в карте, структура которой приведена в табл. 8.2.

Таблица 8.2

Показатели функций			
Номер	Наименование	Оценка реализации	Вес

Структурный контроль СА проводится экспертом для отметки состояний элементов разрабатываемого ПС на ТП и фиксируется в опросной карте контроля СА (табл. 8.3). Результаты оформляются протоколом, передаются в группу контроля и используются для количественных оценок.

Таблица 8.3

Номер вопроса	Содержание	Оценка эксперта
1	Используются ли повторно используемые компоненты (их доля)?	
2	Выделены ли компоненты по функциональному принципу?	
3	Независимы ли компоненты?	
4	Стандартизован ли интерфейс между компонентами?	
5	Предусматриваются ли средства восстановления при ошибках и какие: — использование контрольных точек — контроль данных — другие	
6	Предусматриваются ли средства восстановления при разрушении и какие: — использование страховой копии — восстановление с помощью транзакций — другие	
7	Предусмотрены ли средства защиты данных?	
8	Унифицированы ли основные компоненты?	

Структурный контроль программных компонент до осуществления процесса программирования предназначен для выявления и устранения ошибок в ПТД. Основными контролируемыми показателями являются: полнота СА; технологичность СА; структурная сложность компонент; структурная сложность модулей; трудоемкость кодирования.

Результаты контроля оформляются в сводной карте компоненты (табл. 8.4).

Таблица 8.4

Контроль программной компоненты	Название
Количество проверенных модулей	
обнаруженных ошибок	
модулей с отклонениями от технологии	
требуемых повторных просмотров	
Время контроля	

Карта контроля СА используется при оценке показателя технологичности:

$$T_{\text{са}} = \sum_{i=1}^{N} (W_i \times P_i),$$

где $P_i = \begin{cases} 1, \text{ если ответ положительный,} \\ 0 \text{ — отрицательный;} \end{cases}$ W_i — весовой коэффициент вопроса в карте; N — количество вопросов.

Базовое значение $T_{\text{са}}$ находится в интервале $0{,}8 < T_{\text{са}} \leqslant 1$.

Показатель сложности разрабатываемого ПК определяется с помощью оценочных элементов: количества модулей, их длины, объема и структурной сложностью модуля.

Они определяются по ПТД (паспортов модулей и описаний в СЯВ или ЯПП) и используются при прогнозировании трудоемкости кодирования. При оценке длины, объема, трудоемкости и сложности используются метрики Холстеда [151], в которых длина словаря операндов (K_2) определяется по следующей формуле:

$$K_2 = A \times K_1 + B, \tag{8.34}$$

$$A = M_2/(M_2 \times 2) \times \log 2 \times M_2/2,$$

$$B = M_2 - 2 \times A,$$

где K_1 — длина словаря операторов модуля. Нижняя длина модуля определяется по формуле

$$N = K_1 \times \log(K_1) + K_2 \times \log(K_2), \tag{8.35}$$

а верхняя считается равной $2N$. Запишем объем модуля в битах:

$$V = N \times \log 2\,(K_1 \times K_2). \tag{8.36}$$

Время на программирование модуля можно определить по формуле

$$T = E/(Q \times 60), \tag{8.37}$$

где Q характеризует квалификацию программиста, определяемую по формуле (8.8).

Эти и другие сведения заносятся в карту модуля (табл. 8.5).

В технологичность текста входит качественная оценка структуры модуля, оформления текста в требованиях технологии, а также наличие переменных без индексов и изменение параметра цикла в теле цикла и др.

Таблица 8.5

Карта модуля (имя)	
Характеристики модуля	Обозначения
Длина словаря операторов	K_1
Длина модуля	N
Объем модуля	V
Уровень квалификации	Q
Трудоемкость кодирования	E
Технологичность текста	$T_{\text{са}}$

8.3.2. МЕТОДИКА ТЕСТИРОВАНИЯ ПРОГРАММ НА ПРОЦЕССАХ СБОРКИ И ИСПЫТАНИЙ

Процесс тестирования прикладных программ СОД по соответствующим ТЛ включает следующие основные операции: составление плана тестирования; управление подготовкой и выполнением тестов; анализ результатов тестирования и отладки; повторное тестирование.

В зависимости от того, кто проводит тестирование, рассматриваются три режима:

1) проведение всех видов операций тестирования специальной группой;

2) совместное проведение работ по тестированию разработчиками и группой тестирования, которая планирует, контролирует и принимает решение о доработках;

3) выполнение работ по тестированию самим разработчиком программ на этапах ТЛ.

Первые два режима не всегда выполнимы, как правило, из-за недостатка ресурсов для образования группы тестирования. Поэтому

Таблица 8.6

ТТ	Таблица тестов	Лист
Имя теста	Тестовая ситуация	Ожидаемая реакция

Таблица 8.7

G_1	Журнал регистрации ошибок (статистическая информация)	Лист

Вид объекта

Тестируемый элемент объекта	Количество обнаруженных ошибок	В том числе											
		источник							тип				
		1	2	3	4	5	6	7	8	9	10	11	12

Примечание:

Таблица 8.8

G_2	Журнал регистрации ошибок	Лист

Вид объекта:

Дата	Начало сеанса, час., мин.	Конец сеанса, час., мин.	Номер ошибки	Время выявления	Имя теста	Тип	ЛИСТ	Содержательное описание ошибки	Дата устранения ошибки	Место ошибки

Таблица 8.9

G_3	Журнал регистрации ошибок (сообщение об ошибке)	Лист

Дата обнаружения	ФИО обнаружившего	Дата устранения	Подпись устранившего	Примечание

Описание ситуации:

Реакция на ситуацию:

Предположение об источнике ошибки:

Анализ ошибки и корректирующие действия:

Место ошибки: Тип: Источники:

Описание ошибки:

Описание исправлений:

Описание проверочного теста:

в основном каждый коллектив разработчиков тестирование выполняет сам.

В рамках технологий ТЛ в целях соблюдения единства технологической дисциплины может быть использован набор карт для отображения в них заданий на тестирование отдельных модулей (табл. 8.6) или компонент, собираемых из модулей, и для регистрации ошибочных ситуаций (табл. 8.7—8.9) в ходе тестирования.

Тестирование модулей и компонент состоит в обеспечении на тестах следующих критериев: прохождение каждого оператора не менее одного раза; выполнение программы не менее одного раза на совокупности тестов, включающих наборы входных и выходных данных; тестирование функций хотя бы один раз; тестирование межмодульных интерфейсов. Для обеспечения тестирования технология ТЛ предусматривает реализацию одного из двух методов: пошагового и монолитного [116].

Монолитный метод предполагает выполнение раздельного тестирования каждого модуля с последующим их объединением в программу. Метод пошагового тестирования предполагает, что модули тестируются не изолированно друг от друга, а подключаются поочередно для выполнения тестов к набору уже ранее оттестированных модулей.

При монолитном подходе требуются заглушки и драйверы для каждого модуля. Поскольку при этом подходе модули не интегрируются до самого последнего момента, то это означает, что в течение длительного времени серьезные ошибки в сопряжении могут оставаться необнаруженными. Пошаговый подход основывается на восходящем и нисходящем тестировании.

Восходящее тестирование характеризуется тем, что вначале выполняется автономное тестирование модулей нижнего уровня на основе драйвера без вызова других модулей, затем выбирается очередной модуль, непосредственно вызывающий уже проверенные. Выполняется сборка выбранного модуля с вызываемыми, а затем тестирование в комплексе. Процесс повторяется до тех пор, пока не будет достигнут корневой модуль. При нисходящем подходе вначале выполняется тестирование корневого модуля совместно с заглушками, замещающими непосредственно вызываемые им модули, затем после подсоединения (замещение заглушки) в процессе тестирования выбирается очередной модуль. Процесс продолжается до тех пор, пока не будут подсоединены и протестированы все модули.

Следующим шагом является тщательное тестирование интерфейсов, т. е. проверка правильности вызовов модулей и конкретности передачи экспортируемых и импортируемых значений параметров. Тестирование интерфейсов продолжает тестирование модулей верхних уровней, только вместо заглушек для них используются реальные вызываемые модули. Поэтому многие тесты, которые использовались при автономном тестировании, могут использоваться при тестировании интерфейсов. Новые тесты должны проверять различные ситуации передачи параметров для границ областей, имитировать ошибочные ситуации и т. д.

Монолитное тестирование является более трудоемким, чем пошаговое, так как требует разработки модулей как драйверов, так и за-

глушек. Основным недостатком данного метода тестирования является то, что ошибки в интерфейсах между модулями обнаруживаются позднее, чем при пошаговом тестировании, по которому последствия более серьезны, а стоимость их устранения выше.

После отладки обнаруженных в ходе тестирования ошибок программный объект должен пройти этап повторного тестирования для обнаружения ошибок, появившихся после выполнения корректирующих воздействий.

Завершающими технологическими этапами тестирования объекта являются тестирование функций и комплексные испытания. Оба они преследуют цель определить соответствие полученной программы исходным требованиям и внешним спецификациям. Объектом при тестировании функций выступают отдельные функции или их совокупности, а при комплексных испытаниях — функции в соответствии с ТЗ на ПС.

Основными критериями завершенности тестирования являются: экономические; вероятностные и статистические; структурные (критерии полноты тестирования).

Выбор определенного критерия зависит от классов ПС, реализуемых на ТЛ, от методов тестирования, а также от стадии тестирования. Структурные критерии полноты тестирования наиболее распространены на практике и применяются на разных стадиях тестирования.

Заключение о полноте тестирования производится на основании плана тестирования, таблиц тестов и определяет численную оценку полноты тестирования по формуле

$$P_T = T_п/T, \tag{8.38}$$

где T — общее количество тестовых ситуаций; $T_п$ — количество проверенных ситуаций, соответствующих ожидаемым.

При приемке отлаженных программных модулей, как правило, применяется статистический критерий, основанный на метриках Холстеда [151]: n — словарь, включающий K_1 — число операторов и K_2 — операндов (8.18); N — длина модуля, определяемая по формуле (8.35); V — объем модуля (программы), определяемый по формуле (8.17).

Число ошибок B, оставшихся после отладки в программе, предположительно можно вычислить по формуле

$$B = \frac{V}{3000}, \text{ или } B = \frac{N \log_2 n}{3000}. \tag{8.39}$$

Вероятностные критерии завершенности тестирования применяются при испытаниях ПС для определения надежности и устойчивости функционирования ПС. Применение существующих моделей надежности [154, 179, 182, 184, 189 и др.] позволит достаточно точно прогнозировать надежность ПС в эксплуатации.

Экономические критерии устанавливают соответствие между необходимыми затратами на тестирование и реальными сроками его проведения.

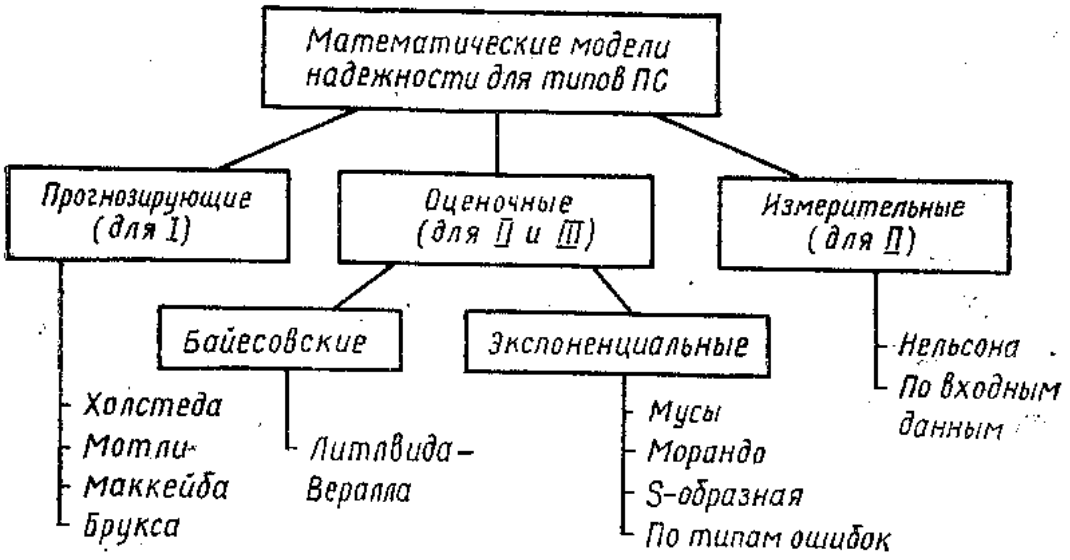


Рис. 8.6. Типы моделей надежности ПС

8.3.3. ОЦЕНКА НАДЕЖНОСТИ НА ЭТАПЕ ИСПЫТАНИЙ И ОПЫТНОЙ ЭКСПЛУАТАЦИИ

На оценку надежности ПС, заключающуюся в определении вероятности безотказной работы ПС в заданных условиях в течение предопределенного периода времени, существенное влияние оказывают два фактора:

программные методы и средства технологии программирования, применяемые на процессах разработки и способствующие достижению требуемой надежности;

тестирование и проверка функционирования созданного ПС со сбором данных о результатах обнаружения ошибок и интенсивности отказов в интервалах времени функционирования.

Методы оценки надежности базируются на аналогичных методах в теории надежности технических средств и отличаются тем, что ошибки в ПС устраняются, улучшая их качество. Отказы в технических средствах, как правило, являются следствием износа и меньше всего зависят от ошибок проектирования (аппаратные средства логически проще, чем программные). Отказы при функционировании технических средств могут привести к непригодности или физическому износу этих средств.

Исследования программных средств на надежность [2, 56, 109, 111, 151, 154, 179, 180, 182, 189, 200 и др.] интенсивно проводились по основным направлениям создания трех типов программных систем.

К первому типу систем относятся программы решения инженерных и научно-технических задач. Они характеризуются неполным жизненным циклом, небольшим объемом и при эксплуатации не требуют всех ресурсов ЭВМ, носят эпизодический и кратковременный характер.

Второй тип представляет собой класс программных систем для информационно-справочных и автоматизированных систем обработки информации, функционирующих вне реального времени.

К третьему типу относятся программы и комплексы, входящие в контур управления и функционирующие в реальном времени, используя все ресурсы ЭВМ (память, быстродействие и др.).

В соответствии с классификацией Хечта [200] в классе этих типов систем сформировались три типа моделей надежности (рис. 8.6).

Прогнозирование надежности — это утверждение, основанное на количественных характеристиках создаваемых на ТП программ (длина программ, объем, число переменных и др.). Модель надежности Холстеда применяется для прогнозирования надежности на ранних этапах разработки (программирование и сборка) и рассмотрена ранее.

Оценочные модели надежности основаны на результатах тестирования, получаемых в тестовой среде при прогонах ПС на тестах и используемых при определении вероятности отказов ПС в реальной операционной среде функционирования. Данные модели исходят из частоты проявления и устранения причины, вызвавшей отказ ПС, поэтому их называют *моделями интенсивности* (сходство с моделями оценки надежности технических средств).

Модели измерения надежности ПС базируются на различных наборах входных данных, определяющих функционирование программ по всему периоду времени. При этом надежность определяется как функция ошибок. Отличительной их особенностью является то, что ПС не модифицируется в течение некоторого периода времени и надежность определяется в момент измерения существующей структуры ПС.

В дальнейшем будем рассматривать оценочные модели, охватывающие более широкий класс программных систем и основанные на истории отказов, зафиксированной в картах регистрации ошибок (см. табл. 8.7—8.9). В картах отмечаются интервалы времени между ошибками либо число ошибок на заданных интервалах.

Для обеспечения надежности рассматриваемого класса ППО СОД будем применять известные оценочные методы надежности, прогнозирующие поведение программ. В процессе эксплуатации они основываются на результатах тестирования и истории отказов, которые фиксируются в журналах регистрации ошибок $G_1 \div G_3$. К используемым на ТЛ моделям оценки надежности ППО относятся модели: Мусы, S-образная, пуассоновского типа, Морандо и др.

Оценочная модель Мусы. Основана на следующих положениях:

тексты адекватно представляют среду функционирования;

происходящие отказы учитываются (оценивается их количество);

интервалы между отказами независимы;

время между отказами распределено по экспоненциальному закону;

интенсивность отказов пропорциональна числу ошибок;

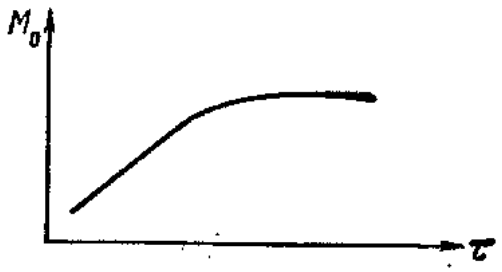
скорость исправления ошибок (относительно времени функционирования) пропорциональна интенсивности их появления.

На основе этих ошибок устанавливаются:

1) зависимость среднего числа отказов от времени функционирования τ

$$m = M_0\left[1 - \exp\left(\frac{c\tau}{M_0 T_0}\right)\right], \tag{8.40}$$

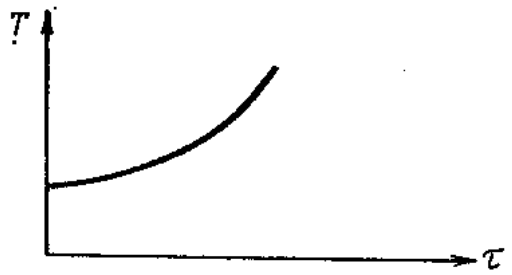
где M_0 — общее число ошибок; T_0 — начальная наработка на отказ; c — коэффициент сжатия тестов (равен времени испытаний).
На графике зависимость выглядит следующим образом:



2) зависимость текущей средней наработки на отказ T от времени функционирования τ

$$T = T_0 \exp\left(\frac{c\tau}{M_0 T_0}\right) \tag{8.41}$$

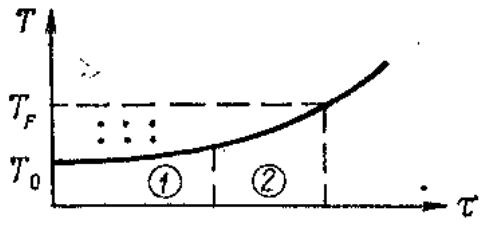
на графике имеет вид



Из уравнений (8.40) и (8.41) следует, что

$$T = \frac{T_0 M_0}{M_0 - m}. \tag{8.42}$$

График этой зависимости представлен областью ①, для которой $M_1 = 1, 2, \ldots$ — номера наблюдений, а $\tau_1, \tau_2, \ldots, \tau_{M_1}$ — время между отказами. Область ② соответствует достижению средней наработки T_p на отказ за время $\Delta\tau$.



Далее по собранным в карте G_2 (см. табл. 8.6) данным об ошибках оцениваются параметры T_0 и M_0. Дополнительное число ошибок, ко-

торые необходимо обнаружить, можно определить по формуле

$$\Delta m = M_0 T_0 \left[\frac{1}{T} - \frac{1}{T_0}\right].$$

В Байесовских моделях интенсивность отказов рассматривается как случайный процесс, зависящий от происшедших отказов, и является суммой случайных процессов:

$$\lambda_i = V_1 + V_2 + \ldots, V_{N-i+1}, \tag{8,43}$$

где $i - 1$ — количество, обнаруженных ошибок к текущему моменту; N — число ошибок.

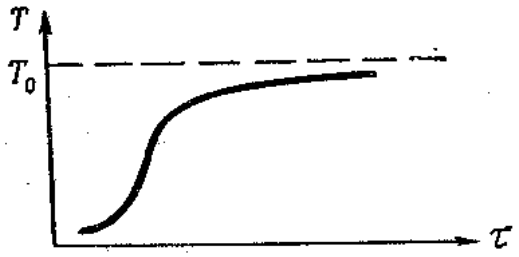
Оценочная S-образная модель. В отличие от модели Мусы здесь зависимость числа отказов от времени функционирования задается в виде

$$m = M_0 \left[1 - \left(1 + \frac{c\tau}{M_0 T_0}\right) \exp\left(-\frac{c\tau}{M_0 T_0}\right)\right]. \tag{8.44}$$

Зависимость текущей средней наработки на отказ от времени функционирования определяется по формуле

$$T = T_0 \exp\left(\frac{c\tau}{M_0 T_0}\right) \tag{8.45}$$

и имеет вид



Из соотношений (8.44) и (8.45) получаем

$$m = M_0 \left[1 - \left(1 - \ln\frac{T}{T_0}\right)\frac{T_0}{T}\right]. \tag{8.46}$$

На основании приведенных формул и собранных в G_2 данных результатов тестирования программного объекта можно получить оценку параметров M_0 и T_0.

Модель, основанная на природе ошибок. Описывается неоднородным Пуассоновским процессом

$$P_r = \{M(t) - n\} = \frac{\{M(t)\}^n}{n!} \exp[-H(t)], \; t \geqslant 0, \tag{8.47}$$

где $M(t)$, $t \geqslant 0$ — общее число ошибок, обнаруженных во временно́м интервале $[0, t]$; $H(t)$ — функция математического ожидания вида

$$H(t) = m_p(t) = a \sum_{i=1}^{k} P_i [1 - \exp(-b_i t)] \tag{8.48}$$

при условиях

$$a > 0, \; 0 < b_k < b_{k-1} < \cdots < b_1 < 1,$$

$$\sum P_i = 1,\ 0 < P_i < 1\ \ (i = \overline{1, k}),$$

где a — число ожидаемых ошибок, существующих в ПС на момент начала тестирования; b_i — интенсивность обнаружения ошибок типа i; k — количество типов ошибок; P_i — доля ошибок типа i.

Ожидаемое число необнаруженных ошибок r_p в момент времени t из формулы (8.48) будет

$$r_p(t) = a \sum_{i=1}^{k} P_i e^{-b_i t_i}. \qquad (8.49)$$

Используя эту модель, можно по полученным фактическим данным об ошибках на процессе тестирования (типы ошибок и интенсивность их обнаружений) рассчитать ожидаемое число необнаруженных ошибок.

Средства автоматизации моделирования надежности ПС. Исходя из рассмотренных методов оценки надежности ПС соответствующие программные средства рассматриваются в виде отдельного технологического модуля — инструментального комплекса. Реализация базируется на регистрации в процессе тестирования истории отказов (времени и числа ошибок) в картах регистрации ошибок (см. табл. 8.6—8.9). Эти данные являются исходными для работы данного модуля.

Процесс выбора модели и получения оценок измерений надежности по выбранной модели заключается в выполнении действий по определению типа модели: на основе моментов между отказами или на подсчете ошибок.

При выборе типа модели, основанной на моментах между отказами, используются предположения, характеризующие специфику модели: независимость моментов между отказами; вероятность появления каждой ошибки; вложенные ошибки не зависят друг от друга; ошибки устраняются после появления; при коррекции не вносятся новые ошибки.

Модели основанные на подсчете ошибок, имеют следующие предположения: интервалы тестирования не зависят один от другого; тестирование на интервалах однородно; количество выявленных ошибок на пересекающихся интервалах времени независимо.

Процесс выбора модели описывается следующими шагами:

1. Изучение данных об ошибках и построение графика зависимости числа ошибок от времени функционирования.

2. Выбор модели, адекватной собранным на процессе тестирования статистическим данным, и анализ построенного на шаге 1 графика.

3. Оценивание параметров модели на основании собранных данных методом максимального правдоподобия.

4. Получение настроенной модели путем подстановки полученных результатов оценки параметров в выбранную модель.

5. Выполнение согласующего теста по проверке на соответствие критерия согласия Колмогорова—Смирнова. Если он дает отрицательный результат, то выбирается другая модель (т. е. переход на шаг 2).

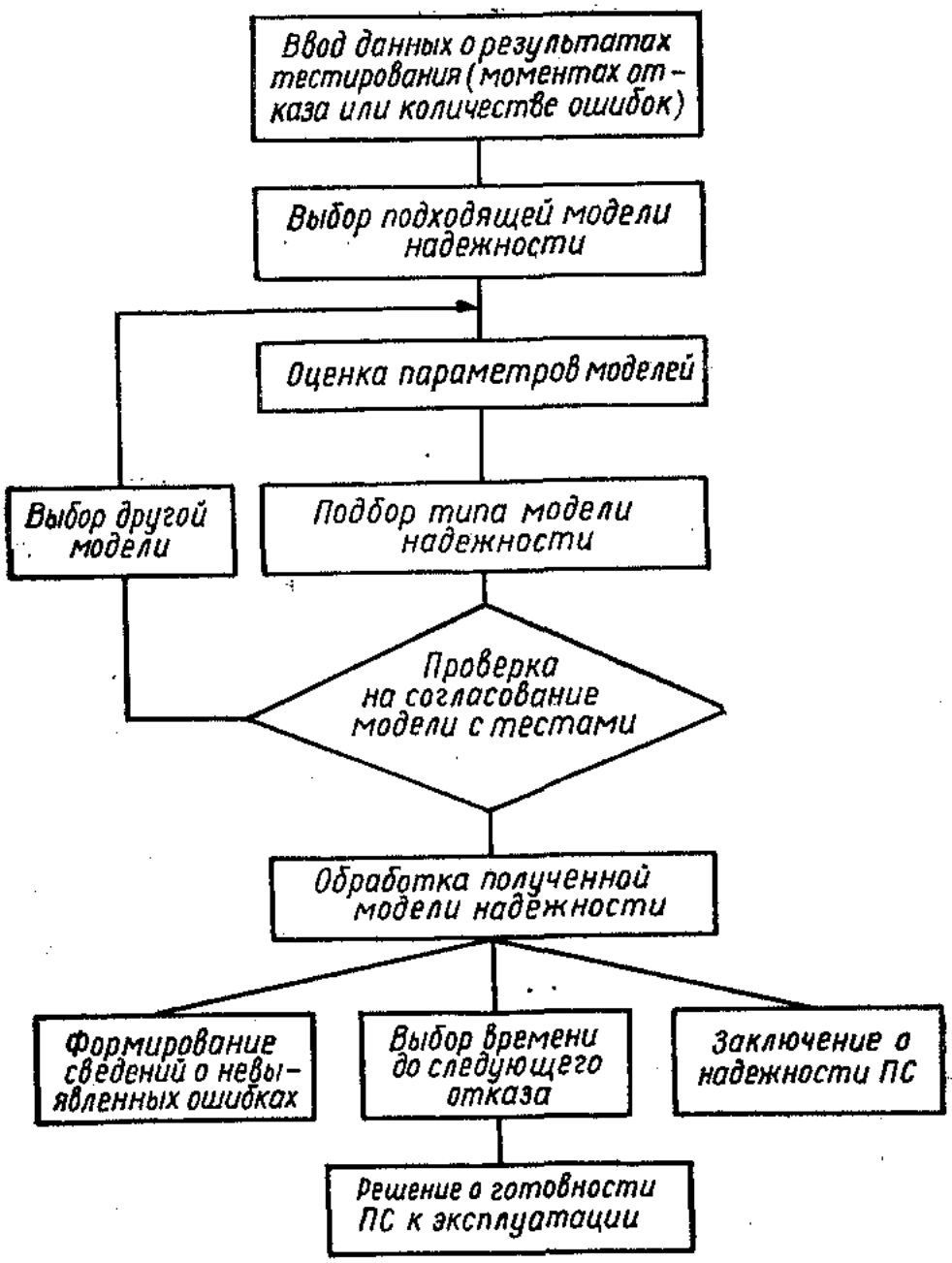


Рис. 8.7. Схема моделирования надежности ПС

6. Получение оценки измерений посредством проведения количественного расчета надежности ПС по выбранной модели (дополнительное время функционирования для достижения заданной наработки на отказ, совокупное число оставшихся ошибок и др.), а также получение доверительных границ для этих оценок.

7. Принятие решения о готовности ПС к эксплуатации путем анализа данных, полученных на шаге 6, и решения о необходимости продолжения тестирования или его завершения.

Общий алгоритм функционирования модуля оценки надежности ПС приведен на рис. 8.7.

ЗАКЛЮЧЕНИЕ

В данной работе изложена методология и инструментальные средства сборочного программирования, разработка которых проводилась с 1976 г. в Институте кибернетики им. В. М. Глушкова.

Подводя итог рассмотренным основам сборочного программирования, отметим очень широкий спектр вопросов, имеющих отношение к данной проблеме. В частности, это относится к вопросам мобильности и совместимости изготовленного ПС с другой средой функционирования. Вместе с тем данный метод входит явно или неявно в состав большинства других подходов к разработке ПС. Примером тому служит наиболее перспективный в настоящее время подход на основе CASE-технологий, используемых для автоматической разработки ПС, начиная с этапа постановки задач предметной области и кончая сопровождением.

Рассмотренные в гл. 6 и 7 технологические аспекты сборочного программирования очень близки следующим основополагающим принципам CASE-технологии:

регламентации всех этапов анализа, проектирования и разработки ПС на единой методологической основе в рамках единой технологии;

смещению основных затрат на разработку с более поздних этапов (программирование, отладка) на более ранние (постановка задач, проектирование) за счет использования формальных моделей Пр0, в рамках которых детализируются описания необходимых задач с последующим автоматическим их переводом в программы.

При решении вопросов генерации программ в CASE-технологии естественно рассматриваются задачи, связанные с проблемами сборочного программирования.

Объем данной книги не позволил более подробно рассмотреть эти и другие вопросы сборочного программирования, каждый из которых заслуживает самостоятельного освещения.

СПИСОК ЛИТЕРАТУРЫ

1. *Абрамович С. М., Букатов А. А.* Определение требований к программным системам: задачи и подход к решению // Технология программирования.— Ростов-на-Дону : РГУ, 1983.— С. 13—24.
2. *Авен О. И., Гурин Н. Н., Коган Я. А.* Оценка качества и оптимизации вычислительных систем.— М. : Наука, 1982.— 464 с.
3. *Агафонов В. Н.* Типы и абстракция данных в языках программирования // Данные в языках программирования.— М. : Мир, 1982.— С. 267—327.
4. *Агафонов В. Н.* Спецификация программ: понятийные средства и их организация.— Новосибирск : Наука, 1987.— 290 с.
5. *Айлиф Дж.* Принципы построения базовой машины.— М. : Мир, 1973.— 120 с.
6. *Александров П. С.* Введение в теорию множеств и общую топологию.— М. : Наука, 1977.— 367 с.
7. *Андерсон Р.* Доказательство правильности программ.— М. ; Мир, 1982.— 164 с.
8. *Андон Ф. И., Захарова Э. Г., Резниченко В. А., Яшунин А. Е.* Интеллектуализация информационных систем // УСиМ.— 1987.— № 1.— С. 84—92.
9. *Ахо А., Хопкрафт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов.— М. : Мир, 1979.— 536 с.
10. *Бабаян Б. А.* Основные принципы программного обеспечения МВК «Эльбрус».— М., 1977.— 11 с.— (Препр. / АН СССР. Ин-т точной механики и вычисл. техники им. С. А. Лебедева; № 7).
11. *Бабенко Л. П.* Проблемно-ориентированные средства языка Кобол.— М. : Статистика, 1979.— 192 с.
12. *Баурн С.* Операционная система UNIX.— М. : Мир, 1986.— 463 с.
13. *Бассакер Р., Саати Т.* Конечные графы и сети.— М. : Наука, 1974.— 236 с.
14. *Бежанова Н. М.* Анализ и систематизация встроенных проблемно-ориентированных систем //УСиМ.— 1981.— № 4.— С. 113—118.
15. *Безбородов Ю. М.* От Фортрана к PL/1.— М. : Наука, 1984.— 208 с.
16. *Барэтисс А. Т.* Структуры данных.— М. : Статистика, 1974.— 408 с.
17. *Боэм Б. У.* Инженерное проектирование программного обеспечения.— М. : Радио и связь, 1985.— 511 с.
18. *Боэм Б., Браун Дж., Каспар Х. и др.* Характеристика качества программного обеспечения : Пер. с англ. Е. К. Масловского.— М. : Мир, 1981.— 208 с.
19. *Брукс Ф. П.* Как проектируются и создаются программные комплексы : мифический человеко-месяц.— М.: Наука, 1979.—321 с.
20. *Бутаков Е. А.* Методы создания качественного программного обеспечения ЭВМ.— М. : Энергоатомиздат, 1984.— 232 с.
21. *Бухштаб Ю. А., Горлин А. И., Камынин С. С. и др.* Интеллектуальный пакет, использующийся при планировании знания о предметной области и функциональных модулях // Изв. АН СССР. Техн. кибернетика.— 1981.— № 5.— С. 113—124.

22. *Брябрин В. Н.* Программное обеспечение персональных ЭВМ.— М. : Наука, 1988.— 272 с.
23. *Вдовкин С. В., Кубенский А. А., Сафонов В. О.* Реализация языка CLU // Прикл. информатика.— 1984.— Вып. 2.— С. 127—130.
24. *Вельбицкий И. В.* Технология программирования.— Киев : Техніка, 1984.— 279 с.
25. *Вельбицкий И. В., Ходаковский В. Н., Шолмов Л. И.* Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6.— М.: Статистика, 1980.— 264 с.
26. *Вегнер П.* Программирование на языке Ада.— М. : Мир, 1983.— 240 с.
27. *Вирт Н.* Систематическое программирование: Введение.— М. : Мир, 1977.— 188 с.
28. *Вирт Н.* Алгоритм + структуры данных = программы: Пер. с англ.— М. : Мир, 1985.— 406 с.
29. *Вирт Н.* Модула-2 // Алгоритмы и алгоритмические языки. Языки программирования.— М. : Наука, 1985.— С. 3—46.
30. *Вишня А. Т., Лаврищева Е. М., Грищенко В. Н. и др.* Система автоматизации программ АПРОП.— Киев, 1980.— 570 с.— Деп. в РФАП АН УССР 21.09.80, № 5707.
31. *Волин В. Г., Грузман В. А., Синичкин В. И., Эпштейн В. Л.* Автоматизация проектирования систем управления.— М. : Финансы и статистика, 1981.— С. 101—111.
32. *Волховер В. Г., Иванов Л. А.* Производственные методы разработки программ.— М. : Финансы и статистика, 1983.— 208 с.
33. *Гантер В.* Методы управления проектированием программного обеспечения.— М. : Мир, 1981.— 392 с.
34. *Гласс Р.* Руководство по надежному программированию.— М. : Финансы и статистика, 1982.— 256 с.
35. *Глушков В. М.* Основы безбумажной информатики.— М. : Наука, 1982.— 552 с.
36. *Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л.* Алгебра. Языки. Программирование.— Киев : Наук. думка, 1974.— 318 с.
37. *Глушков В. М.* Введение в АСУ.— Киев: Техніка, 1974.— 320 с.
38. *Глушков В. М., Вельбицкий И. В.* Технология программирования и проблемы ее автоматизации // УСиМ.— 1976.— № 6.— С. 75—93.
39. *Глушков В. М.* Фундаментальные исследования и технология программирования // Программирование.— 1980.— № 1.— С. 3—13.
40. *Глушков В. М., Капитонова Ю. В., Летичевский А. А.* О применении метода формализованных технических заданий к проектированию программ обработки структур данных // Там же.— 1978.— № 6.— С. 31—43.
41. *Глушков В. М., Лаврищева Е. М., Стогний А. А. и др.* Система автоматизации производства программ (АПРОП) — Киев : Ин-т кибернетики АН УССР.— 1976.— 134 с.
42. *Грис Д.* Наука программирования / Пер. с англ. под ред. А. П. Ершова.— М. : Мир, 1984.— 416 с.
43. *Громов Г. Р.* Национальные информационные ресурсы: Пробл. пром. эксплуатации.— М. : Наука, 1984.— 291 с.
44. *Громов Г. Р.* Персональные вычисления — новый этап информационной технологии // Микропроцессорные средства и системы.— 1984.— № 1.— С. 37—49.
45. *Грищенко В. Н.* Вопросы комплексирования программных средств // УСиМ.— 1987.— № 1.— С. 68—71.
46. *Грищенко В. Н., Лаврищева Е. М.* О создании межъязыкового интерфейса для ОС ЕС // УСиМ.— 1978.— № 1.— С. 34—41.
47. *Грищенко В. Н., Лаврищева Е. М.* О стандартизированной сборке сложных программ // Технологическое и программное обеспечение АСУ.— Киев : Ин-т кибернетики АН УССР, 1978.— С. 36—42.
48. *Грогоно П.* Программирование на языке Паскаль.— М. : Мир, 1982.— 382 с.
49. *Данные* в языках программирования.— М. : Мир, 1982.— 328 с.
50. *Дал У., Дейкстра Э., Хоор К.* Структурное программирование.— М. : Мир, 1975.— 247 с.

51. *Дойл У.* Табличный профессор СУПЕРКАЛК для персонального компьютера.— М. : Финансы и статистика, 1987.— 320 с.
52. *Дворцин В. Н., Прокудин Г. С.* Изготовление программного обеспечения агрегированием из готовых компонент // УСиМ.— 1984.— № 6.— С. 62—64.
53. *Диковский А. Я.* ФОРМАТ: язык для автоматического синтеза и тестирования программ обработки данных // Синтез, тестирование, верификация и отладка программ.— Рига : Изд-во Латв. ун-та, 1981.— С. 87—88.
54. *Добров А. Д. и др.* Особенности комплексирования программ в системе программирования МВК «Эльбрус» / А. Д. Добров, В. М. Пенковский, М. С. Приказчиков, В. С. Чижова // УСиМ.— 1980.— № 4.— С. 92—96.
55. *Ершов А. П.* Введение в теоретическое программирование.— М. : Наука, 1977.— 288 с.
56. *Ершов А. П.* Два облика программирования // Кибернетика.— 1982.— № 6. С. 122—123.
57. *Ершов А. П.* Опыт интегрального подхода к актуальной проблематике программного обеспечения // Там же.— 1984.— № 3.— С. 11—21.
58. *Жоголев Е. А.* Принципы построения многоязыковой системы модульного программирования // Там же.— 1974.— № 4.— С. 78—83.
59. *Жоголев Е. А.* Технологические основы модульного программирования // Там же.— 1980.— № 2.— С. 44—49.
60. *Задыхайло И. Б., Камынин С. С., Любимский Э. З.* Вопросы конструирования вычислительных машин из блоков повышенной квалификации — М., 1971.— 13 с.— (Препр./ АН СССР. Ин-т прикл. математики им. М. В. Келдыша; № 68).
61. *Зайцев Н. Г.* Общесистемное МО ЭВМ третьего поколения для обработки данных.— М. : Статистика, 1980.— 230 с.
62. *Замулин А. В., Скопин И. Н.* Конструирование базы данных на основе концепции абстрактных типов данных // Программирование.— 1981.— № 5.— С. 38—43.
63. *Замулин А. В.* Типы данных в языках программирования и базах данных.— М. : Наука, 1987.— 152 с.
64. *Землянский А. А.* Метод композиции программ иерархической структуры // Прикл. информатика.— 1984.— Вып. 2.— С. 135—142.
65. *Зинглер К.* Методы проектирования программных систем.— М. : Мир, 1985.— 328 с.
66. *Зелковиц М., Шоу А., Гэннон Дж.* Принципы разработки программного обеспечения.— М. : Мир, 1982.— 368 с.
67. *Иванов А. С.* Язык Си. Предварительное описание // Прикл. информатика.— 1985.— Вып. 1.— С. 68—113.
68. *Инмон У., Фридман Л.* Методология экспертной оценки проектных решений для систем с базами данных.— М. : Финансы и статистика, 1986.— 279 с.
69. *Иванников В. П.* Соглашения о связях и ядро операционной системы суперЭВМ // Кибернетика и вычисл. техника.— 1985.— Вып. 1.— С. 49—52.
70. *Йодон Э.* Структурное проектирование и конструирование программ.— М. : Мир, 1979.— 145 с.
71. *Йенсен К., Вирт Н.* Паскаль. Руководство для пользователей и описание языка.— М. : Финансы и статистика, 1982.— 152 с.
72. *Капитонова Ю. В., Летичевский А. А.* О конструировании математических описаний предметных областей // Кибернетика.— 1988.— № 4.— С. 17—25.
73. *Канторович Л. В.* Перспективы работы в области автоматизации программирования на базе крупноблочной системы // Тр. Ин-та математики им. Стеклова.— 1968.— 196.— С. 5—15.
74. *Кахро М. И., Калья А. П., Тыугу Э. Х.* Инструментальная система программирования ЕС ЭВМ (ПРИЗ).— М. : Финансы и статистика, 1981.— 157 с.
75. *Камынин С. С., Любимский Э. З.* Алгоритмический машинно-ориентированный язык — АЛМО // Алгоритмы и алгоритм. яз.— 1967.— Вып. 1.— С. 5—58.
76. *Коллинз Г., Блей Дж.* Структурные методы разработки систем от стратегического планирования до тестирования.— М. : Финансы и статистика, 1986.— 264 с.

77. *Конструирование* систем программирования обработки данных / Под ред. Е. Л. Ющенко.— М. : Статистика, 1979.— 271 с.
78. *Королев Л. Н.* Структуры ЭВМ и их математическое обеспечение.— М. : Наука, 1978.— 362 с.
79. *Корягин Д. А.* Об одном подходе к проблеме разработки системного обеспечения пакетов программ для задач вычислительной физики // Программирование.— 1982.— № 2.— С. 44—50.
80. *Коваль Г. И., Коротун Т. М., Лаврищева Е. М.* Об одном подходе к решению проблемы межмодульного и технологического интерфейсов // Диалоговые системы: Межотрасл. сб. АН СССР и Минвуза СССР.— 1988.
81. *Курглякова Л. И.* Программное обеспечение высокопроизводительных ЭВМ // Вычисл. техника за рубежом в 1986—1987 гг.— М. : ИТМ и ВТ АН СССР, 1987.— С. 136—167.
82. *Кульба В. В., Маминоков А. Г.* Методы анализа и синтеза оптимальных модульных систем обработки данных // Автоматика и телемеханика.— 1980.— № 11.— С. 81—92.
83. *Куприянов В. П. и др.* Технологическая система ТКП-3 / В. П. Куприянов, А. К. Мочалов, Н. А. Маслобойщиков и др. // Приборы и системы упр.— 1983.— № 4.— С. 21—23.
84. *Катков В. Л., Казан В. Н.* Система тестовой обработки «Стрела» // УСиМ.— 1984.— № 1.— С. 106—111.
85. *Катцан Г.* Язык Фортран-77.— М. : Мир, 1982.— 208 с.
86. *Кон П.* Универсальная алгебра.— М. : Мир, 1968.— 352 с.
87. *Лавров С. С.* Синтез программ // Кибернетика.— 1982.— № 6.— С. 11—16.
88. *Лавров С. С.* Осповные понятия и конструкции языков программирования.— М. : Финансы и статистика, 1982.— 80 с.
89. *Лаврищева Е. М.* Методика построения программных комплексов на основе банка модулей // Разработка математических и технических средств АСУ.— Киев: Ин-т кибернетики АН УССР, 1975.— С. 3—12.
90. *Лаврищева Е. М., Райков Л. Д., Стогний А. А.* Развитие модульного принципа создания программ на ЕС ЭВМ // Разработка математических и технических средств автоматизированных систем.— Киев: Ин-т кибернетики АН УССР, 1977.— С. 3—12.
91. *Лаврищева Е. М.* Модульный принцип конструирования больших программ // Там же.— С. 12—20.
92. *Лаврищева Е. М.* Вопросы объединения разноязыковых модулей в ОС ЕС // Программирование.— 1978.— № 1.— С. 22—27.
93. *Лаврищева Е. М.* Подход к промышленной технологии изготовления больших программ // Перспективы развития в системном и теоретическом программировании: Тр. Всесоюз. симп. Новосибирск, 20—22 марта 1978 г.— Новосибирск: Изд-во СО АН СССР, 1978.— С. 122—127.
94. *Лаврищева Е. М.* Об автоматизированном изготовлении программных агрегатов из равноязыковых модулей // УСиМ.— 1979.— № 5.— С. 54—60.
95. *Лаврищева Е. М.* Методика изготовления программных агрегатов // Кибернетика.— 1980.— № 2.— С. 77—82.
96. *Лаврищева Е. М., Грищенко В. Н.* Связь разноязыковых модулей в ОС ЕС.— М. : Финансы и статистика, 1982.— 127 с.
97. *Лаврищева Е. М.* Технология создания прикладного программного обеспечения в СОД.— Киев : О-во «Знание» УССР, 1983.— 15 с.
98. *Лаврищева Е. М., Хоролец Д. С.* Комплекс АПФОРС — средство автоматизации проектирования ППП // Средства информационного обеспечения АН УССР.— Киев : Ин-т кибернетики им. В. М. Глушкова АН УССР, 1984.— С. 67—74.
99. *Лаврищева Е. М.* Принципы технологической подготовки разработки ППО СОД // Проектирование и разработка пакетов программ.— Киев : Ин-т кибернетики им. В. М. Глушкова АН УССР, 1987.— С. 34—40.
100. *Лаврищева Е. М.* Основы технологической подготовки разработки прикладных программ. СОД.— Киев, 1987.— 29 с.— (Препр./АН УССР. Ин-т кибернетики им. В. М. Глушкова; № 87—5).
101. *Лаврищева Е. М., Хоролец Д. С., Куцаченко Л. И. и др.* Комплекс программных средств, обеспечивающих автоматизированное построение пакетов прикладных программ на основе формализованных спецификаций модулей.—

АПФОРС.— Ереван, 1985.— 220 с.— Деп. в ЕрНУЦ СНПО «Алгоритм» 18.05.85, № 104; Рег. в РФАП АН УССР 25.09.87, № АД0002.
102. *Лаврищева Е. М. и др.* Программно-технологический комплекс ведения разработки прикладного программного обеспечения. Технологические документы / Е. М. Лаврищева, Г. И. Коваль, Т. М. Коротун, Е. И. Моренцов.— Киев, 1988.— 571 с.— Рег. в РФАП АН УССР 30.09.88, № АП0218—И.
103. *Лаврищева Е. М.* Технологическая подготовка и программная инженерия // УСиМ.— 1988.— № 1.— С. 48—52.
104. *Лаврищева Е. М.* Модель процесса разработки программных средств // УСиМ.— 1988.— № 5.— С. 43—46.
105. *Литвинов В. В.* Математическое обеспечение проектирования вычислительных систем и сетей.— Киев: Техніка, 182.— 176 с.
106. *Леман Д., Смит М.* Типы данных // Данные в языках программирования.— М. : Мир, 1982.— С. 196—213.
107. *Лебедев В. Н., Соколов А. П.* Введение в систему программирования ОС ЕС.— М.: Статистика, 1978.— 187 с.
108. *Липаев В. В.* Оценка затрат на разработку программных средств.— М. : Финансы и статистика, 1988.— 225 с.
109. *Липаев В. В.* Надежность программного обеспечения АСУ.— М. : Энергоиздат, 1981.— 240 с.
110. *Массер Д.* Спецификации абстрактных типов данных в системе AFFIRM// Требования и спецификация в разработке программ.— М. : Мир, 1984.— С. 199—222.
111. *Майерс Г.* Надежность программного обеспечения.— М. : Мир, 1980.— 360 с.
112. *Малышев В. М.* Вопросы комплексирования программ в ОС ЕС на языках Ассемблер и ПЛ-1, использующих средства мультизадачности // Прикл. информатика.— 1981.— Вып. 1.— С. 145—155.
113. *Маурер У.* Введение в программирование на языке Лисп.— М. : Мир, 1976.— 104 с.
114. *Марчук Г. И., Котов В. Е.* Модульная развиваемая система (концепция).— Новосибирск, 1978.— 2 ч.— (Препр. / ВЦ АН СССР; 86—87).
115. *Минский М.* Фреймы для представления знаний.— М. : Энергия, 1979.— 191 с.
116. *Михалевич В. С. и др.* Организация вычислений в многопроцессорных вычислительных системах /В. С. Михалевич, Ю. В. Капитонова, А. А. Мышевский, И. Н. Молчанов // Кибернетика.— 1984.— № 3.— С. 1—10.
117. *Опарин Г. А.* САТУРН — метасистема для построения ППП // Разработка ППП.— Новосибирск: Наука, 1982.— С. 130—160.
118. *Органик Э.* Организация системы Интел-432.— М. : Мир, 1987.— 448 с.
119. *Орлов В. Н.* Комплексирование программ в ОС ЕС.— М. : Финансы и статистика, 1986.— 136 с.
120. *Операционная* система ОС РВ СМ ЭВМ: Справ. изд. / Г. А. Егоров, В. Л. Король, И. С. Мостов и др.— М. : Финансы и статистика, 1987.— 271 с.
121. *Оши К., Хьюгз П.* Бухгалтерский учет на микроЭВМ: использование прикладного пакета LOTUS 1-2-3.— М.: Финансы и статистика, 1988.— 255 с.
122. *Парасюк И. Н., Сергиенко И. В.* О некоторых задачах модульного анализа при проектировании пакетов программ // УСиМ.— 1982.— № 4.— С. 73—80.
123. *Парнас Д.* Метод спецификации модулей программного обеспечения // Данные в языках программирования.— М. : Мир, 1982.— С. 9—24.
124. *Пелипенко Н. И.* Организация связей между разноязыковыми программами // УСиМ.— 1981.— № 1.— С. 65—68.
125. *Перевозчикова О. Л., Ющенко Е. Л.* Система диалогового решения задач на ЭВМ.— Киев : Наук. думка, 1986.— 264 с.
126. *Пратт Т.* Языки программирования: разработка и реализация.— М.: Мир, 1979.— 576 с.
127. *Программное* обеспечение персональных ЭВМ / А. А. Стогний, С. А. Ананьевский, Я. И. Барсук и др.— Киев : Наук. думка, 1989.— 368 с.

128. *Представление* и использование знаний / Под ред. Х. Уэно, М.Исидзука.— М.: Мир, 1989.— 220 с.
129. *Программирование* на языке Ассемблера ЕС ЭВМ / З. С. Брич, В. И. Воюш, Г С. Дегтярева, Э. В. Ковалевич.— М. : Финансы и статистика, 1986.— 335 с.
130. *Потапов В. И., Деньщиков А. В., Шкода О. Б.* Работа в системе виртуальных машин ЕС./ Под ред. В. Н. Лебедева.— М. : Финансы и статистика, 1988.— 253 с.
131. *Редько В. Н.* Основания композиционного программирования // Программирование.— 1979.— № 3.— С. 3—13.
132. *Редько В. Н.* Семантические структуры программ // Там же.— 1981.— № 3.— С. 3—19.
133. *Родионов С. Т.* Основные концепции модульности и анализа некоторых направлений развития общей технологии программирования // Там же.— 1980.— № 2.— С. 31—38.
134. *Римский Г. В.* Структура и функционирование системы автоматизации модульного программирования // Там. же.— 1987.— № 5.— С. 36—44.
135. *Риз Ч. Е., Стюарт Дж Д.* Rule Master: система конструирования знаний для построения научных экспертных систем / Искусственный интеллект: применение в химии.— М : Мир, 1988.— С. 33—48.
136. *Романовский И. В.* Алгоритмы решения экстремальных задач.— М. : Наука, 1977.— 352 с.
137. *Сергиенко И. В., Парасюк И. Н., Тукалевская Н. И.* Автоматизированные системы обработки данных.— Киев : Наук. думка, 1976.— 256 с.
138. *Сергиенко И. В., Стукало А. С.* Перспективы промышленной разработки прикладных программных систем.— Киев : О-во «Знание» УССР, 1986.— 16 с.
139. *Симонс Дж.* ЭВМ пятого поколения: компьютеры 90-х годов.— М. : Финансы и статистика, 1985.— 291 с.
140. *Современные* проблемы создания систем проектирования // Приборы, средства автоматизации и системы управления // ЦНИИТЭИ приборостроения.— 1980.— Вып. 5.— 45 с.
141. *Трахтенгерц Э. А.* Программное обеспечение автоматизированных систем управления.— М. : Статистика, 1974.— 288 с.
142. *Тыугу Э. Х.* Концептуальное программирование.— М. : Наука, 1984.— 256 с.
143. *Требования* и спецификации в разработке программ.— М. : Мир, 1984.— 344 с.
144. *Турский В.* Методология программирования.— М. : Мир, 1981.— 265 с.
145. *Филина Л. Н.* Вопросы связи модулей, транслированных с языков ФОРТРАН, ПЛ-1, Ассемблер в ОС ЕС ЭВМ // Программирование.— 1980.— № 3.— С. 39—43.
146. *Фомичев В. С., Пютчлер У.* Промежуточные языки систем программирования // ЭВМ в проектировании и производстве.— Л., 1987.— Вып. 3.— С. 251—270.
147. *Фуксман А. Л.* Технологические аспекты создания программных систем.— М. : Статистика, 1979.— 241 с.
148. *Фишер П.* Братиславская программная система BPS // Вычисл. техника соц. стран.— 1984.— Вып. 15.— С. 62—69.
149. *Шошлеков Н. М., Кочетков В. П.* Что нам дал ПЛЮС? // Тез. докл. юбил. науч. сес. СБНАПА «Интерпрограмма» (София, 10—12 окт. 1987 г.).— София, 1987.— С. 107—115.
150. *Хорн Э., Винклер Ф.* Проектирование модульных программных структур/ Вычисл. техника. соц. стран.— 1987.— Вып. 21.— С. 64—72.
151. *Хоар Ч.* О структурной организации данных // Структурное программирование.— М. : Мир, 1975.— С. 92—197.
152. *Хаузер Д., Хирт В., Хоукинс Б.* Операционная система MS DOS.— М. : Финансы и статистика, 1987.— 168 с.
153. *Холстед М. Х.* Начало науки о программах / Пер. с англ. В. М. Юфы.— М. : Финансы и статистика, 1981.— 201 с.

154. *Холл П.* Вычислительные структуры. Введение в нечисленное программирование.— М.: Мир, 1978.— 216 с.
155. *Abarbanel R. M., Williams M. D.* A relational representation for knowledg-bases//Proc. from the first intern. conf. on expert database systems (Charleston. Sc. USA. april 1—4 1986).— Charleston: Univ. South Carolina, 1986.— P. 252—270.
156. *Batley R.* Human error in Computer System: — Prentice Hall, 1983.— 176 p.
157. *Berry D. C.* The problem of implicit knowledge // Dent. of exp. psychol.— 1987.— 4, N 3.— P. 144—151.
158. *Beech D.* Modularity of computer languages // Software — practice and experience.— 1982.— 12, N 10.— P. 929—958.
159. *Beynon-Davies P.* Software Engineering and Knowledge Engineering: Unhappy Bedfellows // Polytechn. of Wales.— Pontypridd. UK.— 1987.— 3, PT. 4.— P. 9—11.
160. *Bril R. J.* Safe Use of Partial Operations // Informatie (Netherlands), Philips, Eindhoven, Netherlands.— 1987.— 29, N 7/8.— P. 700—703, 706—709.
161. *Burstall R., Lampson B. A.* Kernel language for Abstract Data Types and Modules // Lecture Notes in Comput. Sci.— 1984.— 173.— P. 1—50.
162. *Campbell F.* The portable UCSD p. system // Microprocess and Microsyst.— 1983.— 7, N 8.— P. 394—398.
163. *Campbell J. A.* Knowledge-based computing systems and Software engineering // Comput. Phys. Comm.— 1986.— 41, N 2/3.— P. 285—290.
164. *Chapman P., Seiler H.* Estimating Software development costs using INSIGNT expert system shell and COCMOx knowledge bases // 5 Ann. Int Phoenix Conf. Comput. and Comm., Scottsdale. Ariz., 26—28 March 1986: Conf. Proc. Washington (D. C.). 1986.— P. 586—591.
165. Comm. ACM.— 1986.— 29, N 2.— P. 734—744.
166. *Chen P. P.* The entity relationship model: toward a unified view of data // ACM Trans. Database System.— 1976.— P. 9—36.
167. *Chiang J. C.* Software in the Large // Afips Conf. Proc. Vol. 56. Nation. Comput. Conf. (Chicago. IL. USA, June 15—18 1987).— Chicago, 1987.— P. 473—490.
168. *Danahue P.* On the semantics of data types // SIAM J. Comput.— 1979.— 8, N 4.— P. 546—560.
169. *Ford L.* Artificial intelligence and software engineering: a tutorial introduction to their relationship // Artif. intell. rev. (UK), Dept. of comput. sci., Exeter Univ. UK.— 1987.— 1, N 4.— P. 255—273.
170. *Lubars M. D., Harandi M. J.* Intelligent support for software application and design // IEEE expert.— 1986.— N 4.— P. 33—41.
171. *Genesereth M. R., Ginsberg M. Z.* Logic programming // Ibid.— N 9.— P. 933—941.
172. *Goel A. L.* Software reliability models: assumptions limitations and applicability // IEEE Trans. Software Engrg.— 1985.— Vol. SE, N 11/12.— P. 23—45.
173. *Harders N.* Software Engineering as a Methodology // Electronic (West Germany).— 1987.— 36, N 21.— P. 160—163.
174. *Horowitz E., Munson B.* An Expensive View of reusable Software // IEEE Trans. Software.— 1984.— 2, N 5.— P. 477—487.
175. *Macleish K. S., Vennergrund D. A.* An expert system development life cycle model and its relevance to traditional software systems // 5 Ann. Int. Phoenix. Conf. Comput. and Comm. (Scottsdale, Ariz. March 26—28 1986: Conf. Proc.) — Washington (D. C.), 1986.— P. 592—596.
176. *Mohanty S. N.* Software cost Estimation: Present and Future, P. E. 1981.— 1.— P. 27—31.
177. *McCabe J. L.* A complexity measure // IEEE Trans. Software Engrg.— 1976. 2, N 4.— P. 308—320.
178. *Mclauchlan M. R.* Specification of VLSI Designs: A Software Engineering Approach // IEE Colloquium on'VLSI System Design: Specification and Synthesis' (London, UK, Oct. 29 1987).— London, IEE, 1987.— P. 32.

179. *Misra P. N.* Software reliability analysis // IBM systems.— 1983.— 22, N 9.— P. 262—270.
180. *Moranda P. B.* Predictions of software reliability during debugging // Proc. Reliability and Maint Symp. (Washington, June 1975).— Washington (D. C.). 1975.— P. 327—332.
181. *Muller K.* Fundamental Principle of Software Engineering: The Phases Model // Electronic (West Germany).— 1987.— 36, N 21.— P. 169—172.
182. *Musa S. D.* Validity of execution-time theory of software reliability // IEEE Trans. Reliability.— 1979.— 28, N 3.— P. 181—191.
183. *Nakajima R.* An approach to hierarchical and modular construction and verification // Programm. System Comput. Sci. and Technol. — 1982.— P. 126—150.
184. *Okumoto K.* A statistical method for Software Quality Control // IEEE Trans. on Software Engrg.— 11, N 12.— 1984.— P. 1424—1430.
185. *Parkinson R.* Emerging Trends in Software Engineering Tools // Tektronix UK LTD, Marlow, UK, Electron. Prod. des.— 1987.— 7.— P. 47—50.
186. *Penedo M. H., Berry D. M.* The use of a module interconnection specification capability in the Sara System design methodology // Comput. Sci. dept. UCLA.—1978.
187. *Rine D. C.* A method for increasing software productivity called object-oriented design with applications for AL // George Mason Univ. Zairfax. VA. USA.— Afips Conf. Proc. Nation. Comput. Conf. (Chicago JL USA, June 15—18 1987).— Chicago, 1987.— 56.— P. 432—441.
188. *Ross D. T., Shoman K. E.* Structured analysis for requirements definition // IEEE Trans. Software Engrg.— 1971.— Vol. SE, N 3.— P. 6—15.
189. *Jamada S., Ohba M., Osaki S.* S-Shaped reliability Grouth Modeling for Software Error Detections // IEEE Trans. Reliability.— 1983.— R 32, N 5.— P. 475—484.
190. *Shindler M.* Technology forecast. Software // Electron. Design.— 1981.— 29, N 1.— P. 190—199.
191. *Shooman M. L.* Software Engineering: Reliability, Development and Management // Prentice Hall. N. Y.— 1983.— 672 p.
192. *Stenbing H. G.* A software engineering environment for weapon system software // IEEE Trans. Reliability.— 1979.— 28, N 3.— P. 199—204.
193. *Szentes J.* Qualigraph — a software tool for Quality metrics and Graphic documentation // Proc. ESA Estec. software seminar. — NOORDWISN. — 1983.— P. 73—81.
194. *Tichy.* Software development Control Based on Module interconnection // Proc. 4-th Int. Conf. on Software Engrg.— Munchen, 1979.— P. 565—577.
195. *Ueno H.* Expert systems // Dept. system. engrg.— Tokyo: Denki Univ., Japan.— 1987.— 28, N 2.— P. 147—157.
196. *Urban S.* Building intelligence into software tools // IEEE expert.— 1986.— N 1.— P. 1—21.
197. *Waldon S., Gow D., Meystel A.* Updating and organizing world knowledge for a autonomous control system // IEEE Intern. symp. intell. control, Drexel Univ. (Philadelphia, PA, USA, Jan 19—20 1987).— Washington: IEEE Comput. Soc. Press.— 1987.— P. 423—430.
198. *Wegner P.* Capital — intensive software technology // IEEE Trans. Software. Engrg.— 1984.— 1, N 4.— P. 7—45.
199. *Wegner P.* Capital Intensive Technology // Ibid.— 1984.— P. 21—30.
200. *Troy R., Miawad.* Assessment of Software Reliability Models // Ibid.— 1986.— 11, N 9.— P. 25—31.
201. *Weiss D. M.* Teaching a Software Design Metodology // Ibid.— 1987.— Vol. SE — 13, N 11.— P. 1156—1163.
202. *Wollis R. R.* AIDES. Computer Aided Design of Software System. // Software Engrg. Environment.— 1987.— P. 253—256.
203. *Wels J., Mc Keag M.* Structured System Programming.— Prentice-Hall.— 1980.
204. *Wegbreit B.* The Treatment of Data Types in ELI // Comm. ACM.— 1974.— 17, N 5.— P. 251—264.
205. *Wirth N.* LILITH: A Personal Computer for Software Engineer. // Lecture Notes in Computer Science.— 1982.— N 126.— P. 349—397.

206. *The Journal* of Systems and Software.— 1986.— 6, N 4.— P. 307—334.
207. *IEEE Software.*— 1986.— 3, 5.— N 3.— P. 74—75.
208. *Schneidewind N. F.* Application of Program Graphs and Complexity Analysis to Software Development and Testing // IEEE Trans on Reliability.— Vol. R — 28.— P. 192—198.
209. *Szentes J.* QUALIGRAPH: A Software Tool for Quality Metrics and Graphic Documentation: Proc. ESA ESTEC Software Seminar // Noordwijk.— 1983.— October.— P. 73—81.
210. *Johnsson R. K., Wick J. D.* An Overview of the MESA Processor Architecture // Computer Architecture News.— 1982.— N 2.— P. 20—29.

ОГЛАВЛЕНИЕ

Научное издание

ЛАВРИЩЕВА Екатерина Михайловна
ГРИЩЕНКО Владимир Николаевич

СБОРОЧНОЕ ПРОГРАММИРОВАНИЕ

Художественный редактор *И. П. Антонюк*
Технический редактор *А. М. Капустина*
Корректоры *Л. И. Лембак, Л. М. Тищенко*

ИБ № 11543

Сдано в набор 28.02.91. Подп. в печ. 30.08.91. Формат 60×90/16. Бум. тип. № 2. Лит. гарн. Выс. печ. Усл. печ. л. 13,5. Усл. кр-отт. 13,5. Уч.-изд. л. 15,09. Тираж 1190 экз. Заказ № 1-80. Цена 3 р. 70 к.

Издательство «Наукова думка». 252601 Киев, 4, ул. Репина, 3.

Отпечатано с матриц Книжной фабрики им. М. В. Фрунзе, 310057, Харьков-57, Донец-Захаржевского, 6/8 в Нестеровской городской типографии 292310, Нестеров, Львовской обл., ул. Горького, 8. Зак. 2591

НОВЫЕ КНИГИ ИЗДАТЕЛЬСТВА «НАУКОВА ДУМКА»

Кокорева Л. В.,
Перевозчикова О. Л., Ющенко Е. Л.

ДИАЛОГОВЫЕ СИСТЕМЫ И ПРЕДСТАВЛЕНИЕ ЗНАНИЙ:

Справочное пособие.— 1992.— 55 л.— Ил.— 7 р.
Объявлено в темплане 1992 г., поз. 189.

В книге систематизированы методы и средства создания диалоговых систем — основы новых информационных технологий. Для двух классов систем: баз данных и систем знаний — обсуждаются вопросы реализации интеллектуального и дружественного интерфейса с конечными пользователями. Рассмотрены традиционные вопросы о видах языков общения и формах заполнения экранов дисплеев; введены модели представления знаний в диалоговых системах для автоматизированного проектирования систем на всех этапах жизненного цикла.

Применение методов и средств организации диалога иллюстрируются примерами реализации ряда отечественных диалоговых систем, широко апробированных на нескольких поколениях ЭВМ для решения задач из предметных областей САПР, АСНИ, АСУП.

Для специалистов в области управления, информационных систем и программного обеспечения ЭВМ.

Редько В. Н.,
Сергиенко И. В., Стукало А. С.

ПРИКЛАДНЫЕ ПРОГРАММНЫЕ СИСТЕМЫ. АРХИТЕКТУРА, ПОСТРОЕНИЕ, РАЗВИТИЕ.—
1992.— 20 л.—Ил.— 4 р. 30 к.
Объявлено в темплане 1992 г., поз. 191.

В монографии изложены основы теории и построения современных прикладных программных систем (ППС) различных классов и поколений. Приведен анализ концепций развития, методологий, архитектур и методов построения ППС типа развитых, адаптивных и интегрированных ППП, ЭС, систем поддержки принятия решений (СППР) и гибридных ППС как современных, так и новых поколений. Описан ряд оригинальных ППП для решения классов задач оптимизации, моделирования, принятия решений, обработки данных и знаний на ЕС ЭВМ и ПЭВМ. Изложены архитектура и методы построения интегрированных и интеллектуальных ППП, реализующих новые технологии решения задач на ЭВМ конечными пользователями. Рассмотрены методы и инструментарий разработки ППС, интегрированные системы поддержки разработки ППС как перспективное направление индустрии ППС.

Приведены модели и методы построения современных автоматизированных систем с архитектурой открытых многоуровневых интегрированных систем на базе интеграции ППП, СППР, ЭС и других ПС. Рассмотрены новые поколения ППС и перспективы их развития.

Для научных и инженерно-технических работников, занимающихся вопросами разработки и применения программного обеспечения вычислительных машин, систем и сетей, аспирантов, студентов.

Бакаев А. А.,
Гриценко В. И., Козлов Д. Н.

ЭКСПЕРТНЫЕ СИСТЕМЫ
И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ.—
1992.— 15 л.— Ил.— 3 р. 30 к.

Объявлено в темплане 1992 г., поз. 186.

В монографии изложены принципы создания экспертных систем, основы логического программирования и подходы, применяемые при разработке экспертных систем на языке программирования Пролог. Приведен перечень компонентов, входящих в состав экспертной системы, дан анализ требований, предъявляемых к каждому компоненту, указаны особенности его разработки и эксплуатации. Рассмотрены методы организации баз знаний экспертной системы. Дана сравнительная характеристика основных способов логического вывода, применяемых для решения задач. Изложено введение в клаузуальную форму логики предикатов первого порядка и в логическое программирование на языке этой логики. Рассмотрены синтаксис языка программирования Пролог и встроенные предикаты языка. Основные приемы программирования обсуждаются на многочисленных примерах.

Для научных и инженерно-технических работников, занимающихся разработкой экспертных систем и систем управления базами знаний, пользователей таких систем, а также специалистов, работающих в области искусственного интеллекта. Будет полезна студентам соответствующих специальностей.